

DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBB	UUU	UUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBB	UUU	UUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBB	UUU	UUU	GGGGGGGGGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG

[illegible]

```
1 0001 0 MODULE DBGLEVEL3 (IDENT = 'V04-000') =
2 0002 0
3 0003 1 BEGIN
4 0004 1
5 0005 1 *****
6 0006 1 *
7 0007 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
8 0008 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
9 0009 1 * ALL RIGHTS RESERVED.
10 0010 1 *
11 0011 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12 0012 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13 0013 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
14 0014 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15 0015 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16 0016 1 * TRANSFERRED.
17 0017 1 *
18 0018 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19 0019 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20 0020 1 * CORPORATION.
21 0021 1 *
22 0022 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23 0023 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24 0024 1 *
25 0025 1 *****
26 0026 1
27 0027 1
28 0028 1 WRITTEN BY
29 0029 1 John Francis August, 1982
30 0030 1
31 0031 1 MODULE FUNCTION
32 0032 1 This module contains the DEBUG kernel code for performing the
33 0033 1 EVALUATE, EXAMINE and DEPOSIT commands.
34 0034 1
35 0035 1
36 0036 1 REQUIRE 'SRC$:DBGPROLOG.REQ';
37 0170 1
38 0171 1 LIBRARY 'LIB$:DBGGEN.L32';
39 0172 1
40 0173 1 FORWARD ROUTINE
41 0174 1 DBG$COLLECT: NOVALUE,
42 0175 1 DEPOSIT HANDLER,
43 0176 1 DBG$DEPOSIT: NOVALUE,
44 0177 1 DBG$EVALUATE: NOVALUE,
45 0178 1 DBG$EXAMINE: NOVALUE,
46 0179 1 DBG$NEXTLOC,
47 0180 1 DBG$PREVLOC,
48 0181 1 MODIFY PRIMARY,
49 0182 1 PRIMARY ORDER,
50 0183 1 CHECK TEXT DESCRIPTOR,
51 0184 1 FIX_UP_LENGTH;
```



```
53 0185 1 EXTERNAL
54 0186 1     DBG$GL_CURRENT_PRIMARY,      | Pointer to the primary being processed
55 0187 1     DBG$REG_VALUES: VECTOR[,LONG], | Vector of user register values in the
56 0188 1     DBG$GL_CURLOC_VMSDESC,      | Override type for %CURLOC
57 0189 1     DBG$GL_DEPOSIT_TOKEN,      | Assignment operator token
58 0190 1     DBG$GL_IDENTITY_TOKEN,     | Identity operator token
59 0191 1     DBG$GL_DFLTTP,             | Default type from "SET TYPE"
60 0192 1     DBG$GW_DFLTLENG: WORD,      | Length of default data-type
61 0193 1     DBG$GL_SIGN_FLAG;          | Print '+' before signed variable
62 0194 1
63 0195 1 EXTERNAL ROUTINE
64 0196 1     DBG$BUILD_PRIMARY_SUBNODE: NOVALUE, | Add new primary sub-node
65 0197 1     DBG$DATA_LENGTH,           | Get length of a data-item
66 0198 1     DBG$DO_MAPPING,            | Perform "type mapping"
67 0199 1     DBG$EVAL_LANG_OPERATOR,    | Evaluate operator expressions
68 0200 1     DBG$FLUSHBUF: NOVALUE,     | Initialize a new print line
69 0201 1     DBG$GET_TEMPMEM,           | Allocate temporary memory
70 0202 1     DBG$IS_IT_ENTRY,           | Check for CALL entry-mask address
71 0203 1     DBG$INS_DECODE,           | Get length of instruction
72 0204 1     DBG$MAKE_VAL_DESC,         | Construct value descriptor
73 0205 1     DBG$NGET_PAGES,           | Construct page list
74 0206 1     DBG$PC_TO_LINE_LOOKUP,     | Get line & statement number
75 0207 1     DBG$PC_TO_SYMB,           | Look up address in SAT
76 0208 1     DBG$PRIM_TO_VAL,           | Obtain value of data-item
77 0209 1     DBG$PRINT: NOVALUE,        | Formats an output line.
78 0210 1     DBG$PRINT_AGGREGATE : NOVALUE, | Print array or record
79 0211 1     DBG$PRINT_FIELD_REF : NOVALUE, | Print <p,s,e> information
80 0212 1     DBG$PRINT_IDENTIFIER,      | Print name of data-item
81 0213 1     DBG$PRINT_VALUE: NOVALUE,   | Print value in a given radix
82 0214 1     DBG$PRINT_VALUE_AS_INTEGER: NOVALUE, | Print absolute address
83 0215 1     DBG$PUSH_TEMPMEM,          | Save temporary memory state
84 0216 1     DBG$POP_TEMPMEM: NOVALUE,   | Restore
85 0217 1     DBG$NEWLINE: NOVALUE,      | Outputs the output buffer.
86 0218 1     DBG$SAVE_LOC: NOVALUE,     | Save dot
87 0219 1     DBG$SAVE_VAL: NOVALUE,     | Save backslash
88 0220 1     DBG$SET_PAGE_PROT,         | Set page protections
89 0221 1     DBG$SRC_TYPE_PC_SOURCE: NOVALUE, | Type source text
90 0222 1     DBG$STA_ADDRESS_TO_REGDESCR, | Translate address to reg descr
91 0223 1     DBG$STA_REGISTER_NAME,      | Obtain reg name from reg descr
92 0224 1     DBG$STA_SETREGISTERS: NOVALUE, | Store context register values
93 0225 1     DBG$STA_SETCONTEXT: NOVALUE, | Set up context correctly
94 0226 1     DBG$STA_SYMKIND: NOVALUE,   | Get KIND of data item
95 0227 1     DBG$STA_SYMNAME: NOVALUE,   | Get NAME of data item
96 0228 1     DBG$STA_SYMSIZE: NOVALUE,   | Get SIZE of data item
97 0229 1     DBG$STA_SYMTYPE: NOVALUE,   | Get TYPE of data item
98 0230 1     DBG$STA_TYP_RECORD: NOVALUE, | Get symbol table information
99 0231 1     DBG$STA_VARIANT_SELECT,     | Get entry from variant set
100 0232 1     DBG$TYPEID_FOR_ATOMIC,     | Make dummy RST entry
101 0233 1     DBG$UPDATE_WATCHPOINTS: NOVALUE, | Update watched values after DEPOSIT
102 0234 1     LIB$SIGNAL;                | Signal an error
103 0235 1
104 0236 1 LITERAL
105 0237 1
106 0238 1     | Verb codes for the EVALUATE command.
107 0239 1
108 0240 1     EVALUATE                     = 1,      | EVALUATE verb code
109 0241 1     EVALUATE_ADDR               = 2,      | EVALUATE/ADDRESS verb code
```

```

: 110      0242 1    EVALUATE_COND      = 3,    ! EVALUATE/CONDITION verb code
: 111      0243 1
: 112      0244 1
: 113      0245 1
: 114      0246 1    ! Verb codes for the EXAMINE command.
: 115      0247 1
: 116      0248 1    EXAMINE              = 1,    ! EXAMINE verb code
: 117      0249 1    EXAMINE_INSTRUCTION  = 2,    ! EXAMINE/INSTRUCTION verb code
: 118      0250 1    EXAMINE_REGISTER     = 3,    ! EXAMINE register verb code
: 119      0251 1    EXAMINE_SOURCE       = 4,    ! EXAMINE/SOURCE verb code
: 120      0252 1    EXAMINE_CONDITION_VALUE = 5,    ! EXAMINE/CONDITION verb code
: 121      0253 1    EXAMINE_PSL          = 6,    ! EXAMINE the PSL verb code
: 122      0254 1    EXAMINE_PSW          = 7,    ! EXAMINE the PSW verb code
: 123      0255 1
: 124      0256 1    OWN
: 125      0257 1    PAGE_LIST;           ! Pointer to list of pages whose protec-
: 126      0258 1                        ! tion we may have changed

```

```
128 0259 1 GLOBAL ROUTINE DBG$CHANGE_DTYPE(PRM_DESC, NEW_TYPE, NEW_SIZE) =
129 0260 1
130 0261 1 FUNCTION
131 0262 1 -----
132 0263 1
133 0264 1 INPUTS
134 0265 1 -----
135 0266 1
136 0267 1 OUTPUTS
137 0268 1 -----
138 0269 1
139 0270 1
140 0271 2 BEGIN
141 0272 2
142 0273 2 MAP
143 0274 2 PRM_DESC: REF DBG$PRIMARY; ! Pointer to Primary Descriptor
144 0275 2
145 0276 2 LOCAL
146 0277 2 ADDR, ! The current instruction address
147 0278 2 SIZE, ! Size of V-Value Descriptor header
148 0279 2 VAL_DESC: REF DBG$VALDESC; ! Pointer to Value Descriptor
149 0280 2
150 0281 2
151 0282 2 ! Determine what kind of descriptor we have.
152 0283 2 !
153 0284 2 SELECTONE .PRM_DESC[DBG$B_DHDR_TYPE] OF
154 0285 2 SET
155 0286 2
156 0287 2
157 0288 2 ! Handle Primary Descriptors.
158 0289 2 !
159 0290 2 [DBG$K_PRIMARY_DESC]:
160 0291 2 DBG$PRIM_TO_VAL(.PRM_DESC,DBG$K_V_VALUE_DESC,VAL_DESC);
161 0292 2
162 0293 2
163 0294 2 ! Handle Volatile Value Descriptors.
164 0295 2 !
165 0296 2 [DBG$K_V_VALUE_DESC]:
166 0297 2 BEGIN
167 0298 2 SIZE = .PRM_DESC[DBG$W_DHDR_LENGTH];
168 0299 2 VAL_DESC = DBG$GET_TEMPMEM(7.SIZE + (XUPVAL - 1))/XUPVAL);
169 0300 2 CH$MOVE(.SIZE,.PRM_DESC,.VAL_DESC);
170 0301 2 IF .PRM_DESC[DBG$L_DHDR_SYMIDO] NEQ 0
171 0302 2 THEN
172 0303 2 DBG$STA_SETCONTEXT(.PRM_DESC[DBG$L_DHDR_SYMIDO]);
173 0304 2
174 0305 2
175 0306 2 END;
176 0307 2
177 0308 2 ! Any other descriptor type is invalid so we signal an internal
178 0309 2 ! DEBUG error.
179 0310 2 !
180 0311 2 [OTHERWISE]:
181 0312 2 SIGNAL(DBG$_ILLTYPE);
182 0313 2
183 0314 2
184 0315 2 TES;
```



```

185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241

```

```

0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346
0347
0348
0349
0350
0351
0352
0353
0354
0355
0356
0357
0358
0359
0360
0361
0362
0363
0364
0365
0366
0367
0368
0369
0370
0371
0372

```

```

! If the type is DBG$K_NOTYPE, meaning type instruction, we return now.
! IF .NEW_TYPE EQL DBG$K_NOTYPE THEN RETURN .VAL_DESC;

! If we get here then we are overriding the type information. In this
! case, set the FCODE to "descriptor". Also set the "override" flag.
VAL_DESC[DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
VAL_DESC[DBG$V_DHDR_OVERRIDE] = TRUE;
SELECTONE .NEW_TYPE OF
  SET

! Handle the /ASCII, /ASCIZ, and /ASCIIW qualifiers. These refer to the
! zero-terminated and counted ASCII string types.
[DSC$K_DTYPE_AZ,
 DSC$K_DTYPE_AC,
 DSC$K_DTYPE_VT]:
  BEGIN
    IF (.VAL_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS)
      THEN
        SIGNAL(DBG$_UNALIGNED);

    VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_VS;
    VAL_DESC[DBG$B_VALUE_DTYPE] = .NEW_TYPE;
    VAL_DESC[DBG$W_VALUE_LENGTH] =
      FIX_UP_LENGTH(VAL_DESC[DBG$A_VALUE_VMSDESC]);
  END;

! Handle the /ASCII qualifier (ASCII string via its descriptor).
[DBG$K_DTYPE_AD]:
  BEGIN
    IF NOT CHECK_TEXT_DESCRIPTOR(.VAL_DESC)
      THEN
        SIGNAL(DBG$_DESCNOTSET);
  END;

! Handle the plain ASCII text string data type (the /ASCII qualifier).
[DSC$K_DTYPE_T]:
  BEGIN
    IF .NEW_SIZE NEQ 0
      THEN
        VAL_DESC[DBG$W_VALUE_LENGTH] = .NEW_SIZE
      ELSE
        VAL_DESC[DBG$W_VALUE_LENGTH] = DBG$DATA_LENGTH(
          VAL_DESC[DBG$A_VALUE_VMSDESC])/XBPUNIT;

```

```
242 VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_Z;  
243 VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_T;  
244 END;  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298
```

```
! Handle the /INSTRUCTION qualifier.  
[DSC$K_DTYPE_ZI]:  
  BEGIN  
    VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_Z;  
    ADDR = .VAL_DESC[DBG$L_VALUE_POINTER];  
    IF DBG$IS_IT_ENTRY(.ADDR)  
    THEN  
      BEGIN  
        VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZEM;  
        VAL_DESC[DBG$W_VALUE_LENGTH] = 2;  
      END  
    ELSE  
      BEGIN  
        VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;  
        VAL_DESC[DBG$W_VALUE_LENGTH] =  
          DBG$INS_DECODE(.ADDR, FALSE, FALSE) - .ADDR;  
      END;  
    END;  
  
! Handle the /PACKED qualifier.  
[DSC$K_DTYPE_P]:  
  BEGIN  
    VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_Z;  
    VAL_DESC[DBG$B_VALUE_DTYPE] = .NEW_TYPE;  
    IF .NEW_SIZE NEQ 'XX'0000FFFF'  
    THEN  
      VAL_DESC[DBG$W_VALUE_LENGTH] = .NEW_SIZE  
    ELSE  
      BEGIN  
        FIELD  
          PACKED_FIELDS =  
            SET  
              SIGN_NIBBLE = [0,0,4,0]  
            TES;  
        BIND  
          PACKED_DATA = .VAL_DESC[DBG$L_VALUE_POINTER]:  
            BLOCKVECTOR[16,1,BYTE] FIELD(PACKED_FIELDS);  
  
        INCR I FROM 0 TO 15 DO  
          BEGIN  
            IF .PACKED_DATA[I, SIGN_NIBBLE] GTR 9  
            THEN  
              BEGIN  
                VAL_DESC[DBG$W_VALUE_LENGTH] = (.I*2) + 1;  
                EXITLOOP;  
              END  
            END  
          END  
        END  
      END  
    END  
  END
```



```

299      0430      5
300      0431      5
301      0432      5
302      0433      5
303      0434      5
304      0435      5
305      0436      5
306      0437      5
307      0438      5
308      0439      5
309      0440      5
310      0441      5
311      0442      5
312      0443      5
313      0444      5
314      0445      5
315      0446      5
316      0447      5
317      0448      5
318      0449      5
319      0450      5
320      0451      5

```

```

      END;
      END;
      END;
      END;
      ! Handle any other data type.
      [OTHERWISE]:
      BEGIN
      VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_2;
      VAL_DESC[DBG$B_VALUE_DTYPE] = .NEW_TYPE;
      VAL_DESC[DBG$W_VALUE_LENGTH] = .NEW_SIZE;
      END;
      TES;
      RETURN .VAL_DESC;
      END;

```

```

      .TITLE  DBGLEVEL3
      .IDENT  \V04-000\
      .PSECT  DBG$OWN,NOEXE, PIC,2
00000 PAGE_LIST:
      .BLKB   4
      .EXTRN  DBG$GL_CURRENT_PRIMARY
      .EXTRN  DBG$REG_VALUES, DBG$GL_CURLOC_VMSDESC
      .EXTRN  DBG$GL_DEPOSIT_TOKEN
      .EXTRN  DBG$GL_IDENTITY_TOKEN
      .EXTRN  DBG$GL_DFLTYP, DBG$GW_DFLTLENG
      .EXTRN  DBG$GL_SIGN_FLAG
      .EXTRN  DBG$BUILD_PRIMARY_SUBNODE
      .EXTRN  DBG$DATA_LENGTH
      .EXTRN  DBG$DO_MAPPING, DBG$EVAL_LANG_OPERATOR
      .EXTRN  DBG$FLOSHBUF, DBG$GET_TEMPMEM
      .EXTRN  DBG$IS_IT_ENTRY
      .EXTRN  DBG$INS_DECODE, DBG$MAKE_VAL_DESC
      .EXTRN  DBG$NGET_PAGES, DBG$PC_TO_LINE_LOOKUP
      .EXTRN  DBG$PC_TO_SYMD
      .EXTRN  DBG$PRIM_TO_VAL
      .EXTRN  DBG$PRINT, DBG$PRINT_AGGREGATE
      .EXTRN  DBG$PRINT_FIELD_REF
      .EXTRN  DBG$PRINT_IDENTIFIER
      .EXTRN  DBG$PRINT_VALUE
      .EXTRN  DBG$PRINT_VALUE_AS_INTEGER
      .EXTRN  DBG$PUSH_TEMPMEM
      .EXTRN  DBG$POP_TEMPMEM
      .EXTRN  DBG$NEWLINE, DBG$SAVE_LOC
      .EXTRN  DBG$SAVE_VAL, DBG$SET_PAGE_PROT
      .EXTRN  DBG$SRC_TYPE_PC_SOURCE

```

				.EXTRN		DBG\$STA_ADDRESS TO REGDESCR	
				.EXTRN		DBG\$STA_REGISTER NAME	
				.EXTRN		DBG\$STA_SETREGISTERS	
				.EXTRN		DBG\$STA_SETCONTEXT	
				.EXTRN		DBG\$STA_SYMKIND	
				.EXTRN		DBG\$STA_SYMNAME	
				.EXTRN		DBG\$STA_SYMSIZE	
				.EXTRN		DBG\$STA_SYMTYPE	
				.EXTRN		DBG\$STA_TYP_RECORD	
				.EXTRN		DBG\$STA_VARIANT_SELECT	
				.EXTRN		DBG\$TYPEID_FOR_ATOMIC	
				.EXTRN		DBG\$UPDATE_WATCHPOINTS	
				.EXTRN		LIB\$SIGNAL	
				.PSECT		DBG\$CODE,NOWRT, SHR, PIC,0	
				.ENTRY		DBG\$CHANGE_DTYPE, Save R2,R3,R4,R5,R6,R7	0259
				MOVAB		LIB\$SIGNAL, R7	
				SUBL2		#4, SP	
				MOVL		PRM_DESC, R6	0285
				CMPB		2(R6), #121	0291
				BNEQ		1\$	
				PUSHL		SP	0292
				MOVZBL		#131, -(SP)	
				PUSHL		R6	
				CALLS		#3, DBG\$PRIM_TO_VAL	
				BRB		3\$	
				CMPB		2(R6), #131	0297
				BNEQ		2\$	
				MOVZWL		(R6), SIZE	0299
				MOVAB		3(R2), R0	0300
				DIVL3		#4, R0, -(SP)	
				CALLS		#1, DBG\$GET_TEMPMEM	
				MOVL		R0, VAL_DESC	
				MOVCL3		SIZE, (R6), @VAL_DESC	0301
				TSTL		12(R6)	0302
				BEQL		3\$	
				PUSHL		12(R6)	0304
				CALLS		#1, DBG\$STA_SETCONTEXT	
				BRB		3\$	0285
				PUSHL		#165848	0313
				CALLS		#1, LIB\$SIGNAL	
				MOVL		NEW_TYPE, R4	0320
				CMPL		R4, #128	
				BNEQ		4\$	
				MOVL		VAL_DESC, R0	
				RET			
				MOVL		VAL_DESC, R2	0326
				MOVB		#3, 6(R2)	
				BISB2		#128, 4(R2)	0327
				CMPL		R4, #37	0335
				BLSS		6\$	
				CMPL		R4, #39	
				BGTR		6\$	
				MOVAB		20(R2), R3	0339
				CMPB		3(R3), #13	
				BNEQ		5\$	

57	00000000G	00	00FC	00000	
5E		04	9E	00002	
56		04	C2	00009	
79	8F	02	AC	D0	0000C
			A6	91	00010
			11	12	00015
			5E	DD	00017
	7E	83	8F	9A	00019
			56	DD	0001D
00000000G	00		03	FB	0001F
			3B	11	00026
83	8F	02	A6	91	00028 1\$:
			2B	12	0002D
	52		66	3C	0002F
	50	03	A2	9E	00032
7E	50		04	C7	00036
00000000G	00		01	FB	0003A
	6E		50	D0	00041
00	BE		52	28	00044
		0C	A6	D5	00049
			15	13	0004C
		0C	A6	DD	0004E
00000000G	00		01	FB	00051
			09	11	00058
		000287DB	8F	DD	0005A 2\$:
	67		01	FB	00060
	54	08	AC	D0	00063 3\$:
00000080	8F		54	D1	00067
			04	12	0006E
	50		6E	D0	00070
			04	04	00073
	52		6E	D0	00074 4\$:
06	A2		03	90	00077
04	A2	80	8F	88	0007B
	25		54	D1	00080
			2C	19	00083
	27		54	D1	00085
			27	14	00088
	53	14	A2	9E	0008A
	0D	03	A3	91	0008E
			09	12	00092

			00028D08	8F	DD	00094	PUSHL	#167176	0341	
				01	FB	0009A	CALLS	#1, LIB\$SIGNAL		
03	67			0B	90	0009D	58:	MOVB	#1, 3(R3)	0343
02	A3			54	90	000A1	MOVB	R4, 2(R3)	0344	
				53	DD	000A5	PUSHL	R3	0346	
0000V	CF			01	FB	000A7	CALLS	#1, FIX_UP_LENGTH		
	63			50	B0	000AC	MOVW	R0, (R3)		
				7C	11	000AF	BRB	12\$	0328	
	38			54	D1	000B1	68:	CMPL	R4, #56	0352
				15	12	000B4	BNEQ	7\$		
				52	DD	000B6	PUSHL	R2	0354	
0000V	CF			01	FB	000B8	CALLS	#1, CHECK_TEXT_DESCRIPTOR		
	6D			50	E8	000BD	BLBS	R0, 12\$		
			00028F50	8F	DD	000C0	PUSHL	#167760	0356	
	67			01	FB	000C6	CALLS	#1, LIB\$SIGNAL		
				62	11	000C9	BRB	12\$	0328	
	0E			54	D1	000CB	78:	CMPL	R4, #14	0363
				25	12	000CE	BNEQ	10\$		
	53	14		A2	9E	000D0	MOVAB	20(R2), R3	0367	
		0C		AC	D5	000D4	TSTL	NEW_SIZE	0365	
				06	13	000D7	BEQL	8\$		
	63		0C	AC	B0	000D9	MOVW	NEW_SIZE, (R3)	0367	
				10	11	000DD	BRB	9\$		
				53	DD	000DF	88:	PUSHL	R3	0371
51	00000000G	00		01	FB	000E1	CALLS	#1, DBG\$DATA_LENGTH		
		50		08	C7	000E8	DIVL3	#8, R0, R1		
		63		51	B0	000EC	MOVW	R1, (R3)		
	02	A3		0E	B0	000EF	98:	MOVW	#14, 2(R3)	0374
				72	11	000F3	BRB	18\$	0328	
	16			54	D1	000F5	108:	CMPL	R4, #22	0380
				35	12	000F8	BNEQ	13\$		
	53	14		A2	9E	000FA	MOVAB	20(R2), R3	0382	
		03		A3	94	000FE	CLRB	3(R3)		
	55	18		A2	D0	00101	MOVL	24(R2), ADDR	0383	
				55	DD	00105	PUSHL	ADDR	0384	
00000000G	00			01	FB	00107	CALLS	#1, DBG\$IS_IT_ENTRY		
	09			50	E9	0010E	BLBC	R0, 11\$		
	02	A3		17	90	00111	MOVB	#23, 2(R3)	0387	
		63		02	B0	00115	MOVW	#2, (R3)	0388	
				4D	11	00118	BRB	18\$	0384	
	02	A3		16	90	0011A	118:	MOVB	#22, 2(R3)	0393
				7E	7C	0011E	CLRQ	-(SP)	0395	
				55	DD	00120	PUSHL	ADDR		
00000000G	00			03	FB	00122	CALLS	#3, DBG\$INS_DECODE		
63	50			55	A3	00129	SUBW3	ADDR, R0, (R3)		
				38	11	0012D	128:	BRB	18\$	0328
	15			54	D1	0012F	138:	CMPL	R4, #21	0403
				2A	12	00132	BNEQ	16\$		
	16	A2		54	9B	00134	MOVZBW	R4, 22(R2)	0406	
0000FFFF	8F		0C	AC	D1	00138	CMPL	NEW_SIZE, #65535	0407	
				20	12	00140	BNEQ	17\$		
				50	D4	00142	CLRL	1	0423	
09	18 B240	04		00	ED	00144	148:	CMPZV	#0, #4, @24(R2)[1], #9	0425
				0B	15	0014B	BLEQ	15\$		
	51	50		01	78	0014D	ASHL	#1, 1, R1	0428	
14	A2	51		01	A1	00151	ADDW3	#1, R1, 20(R2)		
				0F	11	00156	BRB	18\$	0427	



DBGLEVEL3  
V04-000

6 7  
16-Sep-1984 01:30:26  
14-Sep-1984 12:17:02

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGLEVEL3.B32;1

Page 10  
(3)

EB		50		0F	F3	00158	158:	AOBLEQ	#15, 1, 148	:	0423
				09	11	0015C		BRB	188	:	0328
	16	A2		54	9B	0015E	168:	MOVZBW	R4, 22(R2)	:	0444
	14	A2	0C	AC	80	00162	178:	MOVW	NEW_SIZE, 20(R2)	:	0445
		50		52	00	00167	188:	MOVL	R2, R0	:	0450
				04	0016A			RET		:	0451

; Routine Size: 363 bytes.    Routine Base: DBGSCODE + 0000

```

322 0452 GLOBAL ROUTINE DBG$COLLECT(PRM_DESC) : NOVALUE =
323 0453
324 0454 FUNCTION
325 0455 -----
326 0456
327 0457 INPUTS
328 0458 -----
329 0459
330 0460 OUTPUTS
331 0461 -----
332 0462
333 0463
334 0464 BEGIN
335 0465
336 0466 MAP
337 0467     PRM_DESC: REF DBG$PRIMARY;      ! Pointer to Primary Descriptor
338 0468
339 0469 BUILTIN
340 0470     REMQUE;                          ! Remove queue entry from list
341 0471
342 0472 LOCAL
343 0473     XXXXXXXX;                       !<-----
344 0474
345 0475
346 0476
347 0477
348 0478
349 0479 IF (.PRM_DESC NEQA 0) THEN
350 0480     IF (.PRM_DESC[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC) THEN
351 0481         IF .PRM_DESC[DBG$V_DHDR_AGGR] THEN
352 0482             BEGIN
353 0483                 LOCAL SUB_NODE : REF DBG$PRIM_NODE;
354 0484
355 0485                 SUB_NODE = .PRM_DESC[DBG$L_PRIM_BLINK];
356 0486
357 0487                 IF (.SUB_NODE[DBG$B_PNODE_FCODE] EQL RST$K_TYPE_ARRAY)
358 0488                     AND
359 0489                     (.SUB_NODE[DBG$B_PNARR_DTYPE] EQL DSC$K_DTYPE_T)
360 0490                     AND
361 0491                     (.SUB_NODE[DBG$W_PNARR_LENGTH] EQL 1)
362 0492             THEN
363 0493                 BEGIN
364 0494                     BIND S_VECTOR = SUB_NODE[DBG$A_PNARR_SVECTOR] : DBG$PRIM_NODE_SUBS;
365 0495                     LOCAL DIMS, SIZE, BASE, TYPEID, SYMID;
366 0496                     DIMS = .SUB_NODE[DBG$B_PNARR_DIMCNT] - 1;
367 0497                     IF .S_VECTOR[DIMS, DBG$L_PNSUB_STRIDE] NEQ 1 THEN RETURN;
368 0498                     IF .S_VECTOR[DIMS, DBG$L_PNSUB_TYPEID] NEQ 0 THEN RETURN;
369 0499
370 0500                     BASE = .S_VECTOR[DIMS, DBG$L_PNSUB_LBOUND];
371 0501                     SIZE = (.S_VECTOR[DIMS, DBG$L_PNSUB_UBOUND] - .BASE) + 1;
372 0502                     PRM_DESC[DBG$W_PRIM_OFFSET] = .BASE;
373 0503                     PRM_DESC[DBG$W_PRIM_LENGTH] = .SIZE;
374 0504                     PRM_DESC[DBG$V_DHDR_SUBREF] = TRUE;
375 0505                     PRM_DESC[DBG$V_DHDR_TMPREF] = TRUE;
376 0506                     TYPEID = DBG$TYPEID_FOR_ATOMIC(DSC$K_DTYPE_T, .SIZE * %BPUNIT, FALSE);
377 0507                     IF .DIMS GTR 0
378 0508                     THEN

```

```

379 0509 BEGIN
380 0510 SUB_NODE[DBG$B_PNARR_DIMCNT] = .DIMS;
381 0511 SUB_NODE[DBG$B_PNARR_CELLTYPE] = .TYPEID;
382 0512 END
383 0513 ELSE
384 0514 BEGIN
385 0515 SYMID = .SUB_NODE[DBG$B_PNODE_SYMID];
386 0516 REMQUE(.SUB_NODE, SUB_NODE);
387 0517 DBG$BUILD_PRIMARY_SUBNODE(.PRM_DESC, RST$K_DATA, SYMID,
388 0518 RST$K_TYPE_ATOMIC, .TYPEID, 0);
389 0519 PRM_DESC[DBG$V_DHDR_AGGR] = FALSE;
390 0520 SUB_NODE = .PRM_DESC[DBG$B_PNARR_DIMCNT];
391 0521 SUB_NODE[DBG$B_PNODE_RELOC] = -.BASE;
392 0522 END;
393 0523 END;
394 0524 END;
395 0525 ! End of dbg$collect

```

			001C 00000	.ENTRY	DBG\$COLLECT, Save R2,R3,R4	0452
	51	04	AC D0 00002	MOVL	PRM_DESC, R1	0479
		01	12 00006	BNEQ	1\$	
			04 00008	RET		
79	8F	02	A1 91 00009 1\$:	CMPB	2(R1), #121	0480
		34	12 0000E	BNEQ	3\$	
	01	04	A1 E8 00010	BLBS	4(R1), 2\$	0481
			04 00014	RET		
	52	18	A1 D0 00015 2\$:	MOVL	24(R1), SUB_NODE	0485
	01	09	A2 91 00019	CMPB	9(SUB_NODE), #1	0487
		25	12 0001D	BNEQ	3\$	
	0E	1A	A2 91 0001F	CMPB	26(SUB_NODE), #14	0489
		1F	12 00023	BNEQ	3\$	
	01	1C	A2 B1 00025	CMPW	28(SUB_NODE), #1	0491
		19	12 00029	BNEQ	3\$	
	53	1B	A2 9A 0002B	MOVZBL	27(SUB_NODE), DIMS	0496
		53	D7 0002F	DECL	DIMS	
50	53		14 C5 00031	MULL3	#20, DIMS, R0	0497
		2C	A240 9F 00035	PUSHAB	44(SUB_NODE)[R0]	
	01		9E D1 00039	CMPB	@(SP)+, #1	
		6D	12 0003C	BNEQ	5\$	
		38	A240 9F 0003E	PUSHAB	56(SUB_NODE)[R0]	0498
		9E	D5 00042	TSTL	@(SP)+	
		65	12 00044 3\$:	BNEQ	5\$	
	54	30	A240 9F 00046	PUSHAB	48(SUB_NODE)[R0]	0500
		9E	D0 0004A	MOVL	@(SP)+, BASE	
50	9E	34	A240 9F 0004D	PUSHAB	52(SUB_NODE)[R0]	0501
		54	C3 00051	SUBL3	BASE, @(SP)+, R0	
		50	D6 00055	INCL	SIZE	
10	A1		54 B0 00057	MOVW	BASE, 16(R1)	0502
12	A1		50 B0 0005B	MOVW	SIZE, 18(R1)	0503
04	A1	0102	8F AB 0005F	BISW2	#258, 4(R1)	0505
		7E	D4 00065	CLRL	-(SP)	0506
7E	50		03 78 00067	ASHL	#3, SIZE, -(SP)	
		0E	DD 0006B	PUSHL	#14	



00000000G	00		03	FB	0006D	CALLS	#3, DBGSTYPEID_FOR_ATOMIC		
			53	D5	00074	TSTL	DIMS	...	0507
			09	15	00076	BLEQ	48		
18	A2		53	90	00078	MOVB	DIMS, 27(SUB_NODE)		0510
24	A2		50	D0	0007C	MOVL	TYPEID, 36(SUB_NODE)		0511
				04	00080	RET			0507
	51	10	A2	D0	00081	MOVL	16(SUB_NODE), SYMID		0515
	52		62	0F	00085	REMQUE	(SUB_NODE), SUB_NODE		0516
			7E	D4	00088	CLRL	-(SPT		0517
			50	DD	0008A	PUSHL	TYPEID		0518
			02	DD	0008C	PUSHL	#2		0517
			51	DD	0008E	PUSHL	SYMID		
			06	DD	00090	PUSHL	#6		
	53	04	AC	D0	00092	MOVL	PRM_DESC, R3		
			53	DD	00096	PUSHL	R3		
00000000G	00		06	FB	00098	CALLS	#6, DBG\$BUILD_PRIMARY_SUBNODE		
04	A3		01	8A	0009F	BICB2	#1, 4(R3)		0519
	52	18	A3	D0	000A3	MOVL	24(R3), SUB_NODE		0520
14	A2		54	CE	000A7	MNEGL	BASE, 20(SUB_NODE)		0521
			04	000AB	55:	RET			0525

; Routine Size: 172 bytes, Routine Base: DBG\$CODE + 016B

```

397 0526 1 ROUTINE DEPOSIT_HANDLER(SIGNAL_ARGS: REF BLOCK[,BYTE]) =
398 0527
399 0528 FUNCTION
400 0529     This routine is the handler for errors that are signalled during
401 0530     the processing of a DEPOSIT command. A handler is necessary so
402 0531     that we can restore page protections that we may have changed.
403 0532
404 0533 INPUTS
405 0534     NONE
406 0535
407 0536 OUTPUTS
408 0537     -----
409 0538
410 0539
411 0540 BEGIN
412 0541
413 0542 LOCAL
414 0543     MESSAGE_VECT;
415 0544
416 0545     ! If we get here the second time around (on the unwind from the
417 0546     ! final handler) then resignal the exception. Do not free up
418 0547     ! the page list again.
419 0548
420 0549     IF .SIGNAL_ARGS[CHFSL_SIG_NAME] EQL SS$_UNWIND
421 0550     THEN
422 0551         RETURN SS$_RESIGNAL;
423 0552
424 0553     IF .PAGE_LIST NEQ 0
425 0554     THEN
426 0555         DBG$SET_PAGE_PROT(PAGE_LIST,TRUE,MESSAGE_VECT);
427 0556
428 0557     RETURN SS$_RESIGNAL;
429 0558     END;

```

```

0004 00000 DEPOSIT_HANDLER:
      52 00000000' EF 9E 00002 .WORD Save R2
      SE 04 C2 00009 MOVAB PAGE_LIST, R2
      50 04 AC D0 0000C SUBL2 #4, SP
00000920 8F 04 A0 D1 00010 MOVL SIGNAL_ARGS, R0
      11 13 00018 CMPL 4(R0),-#2336
      62 D5 0001A BEQL 1$
      0D 13 0001C TSTL PAGE_LIST
      SE DD 0001E BEQL 1$
      01 DD 00020 PUSHL SP
      52 DD 00022 PUSHL #1
00000000G 00 03 FB 00024 PUSHL R2
      50 0918 BF 3C 0002B CALLS #3, DBG$SET_PAGE_PROT
      04 00030 1$: MOVZWL #2328, R0
      RET

```

: Routine Size: 49 bytes, Routine Base: DBG\$CODE + 0217

```

0526
0549
0553
0555
0557
0558

```

```

431 0559 1 GLOBAL ROUTINE DBG$DEPOSIT(VERB_NODE : REF DBG$VERB_NODE) : NOVALUE =
432 0560 1
433 0561 1 FUNCTION
434 0562 1 This routine accepts as input the command execution tree constructed
435 0563 1 by the parse network and performs the semantic actions corresponding to
436 0564 1 the parsed DEPOSIT command. If the command cannot be executed, a message
437 0565 1 argument vector is constructed and returned.
438 0566 1
439 0567 1 Upon entrance to this routine, the command has been classified as plain
440 0568 1 DEPOSIT or instruction DEPOSIT, and all default and override types have
441 0569 1 been set up in the adverb nodes.
442 0570 1
443 0571 1 There should be two noun nodes. The first is the target of the deposit
444 0572 1 while the second represents the source (either a value descriptor or a
445 0573 1 pointer to a counted string for instruction DEPOSITS).
446 0574 1
447 0575 1 INPUTS
448 0576 1 VERB_NODE - A longword containing the address of the verb (head)
449 0577 1 node of the command execution tree
450 0578 1
451 0579 1 OUTPUTS
452 0580 1 -----
453 0581 1
454 0582 1 BEGIN
455 0583 1
456 0584 1 ROUTINE TEXT_LENGTH(VAL_DESC : REF DBG$VALDESC) =
457 0585 1 BEGIN
458 0586 1 LOCAL LENGTH;
459 0587 1 SELECT ONE .VAL_DESC[DBG$B_VALUE_DTYPE] OF
460 0588 1
461 0589 1 SET
462 0590 1 [DSC$K_DTYPE_T]: LENGTH = .VAL_DESC[DBG$W_VALUE_LENGTH];
463 0591 1 [DSC$K_DTYPE_VT]: LENGTH = .(.VAL_DESC[DBG$L_VALUE_POINTER])<0,16,0>;
464 0592 1 [DSC$K_DTYPE_AC]: LENGTH = .(.VAL_DESC[DBG$L_VALUE_POINTER])<0, 8,0>;
465 0593 1 [OTHERWISE]: SIGNAL(DBG$_ILLTYPE);
466 0594 1
467 0595 1 TES;
468 0596 1 RETURN .LENGTH;
469 0597 1 END;

```

```

0004 00000 TEXT_LENGTH:
50 04 AC D0 00002 .WORD Save R2
51 16 A0 9A 00006 MOVL VAL_DESC, R0
0E 51 91 0000A MOVZBL 22(R0), R1
06 12 0000D CMPB R1, #14
52 14 A0 3C 0000F BNEQ 1$
23 11 00013 MOVZWL 20(R0), LENGTH
25 51 91 00015 1$: BRB 4$
06 12 00018 CMPB R1, #37
52 18 B0 3C 0001A BNEQ 2$
18 11 0001E MOVZWL 24(R0), LENGTH
26 51 91 00020 2$: BRB 4$
CMPB R1, #38

```

0586  
0589  
0591  
0592  
0593



```

          52      18      06 12 00023      BNEQ      3$
          80 9A 00025      MOVZBL 224(R0), LENGTH
          0D 11 00029      BRB      4$
00000000G 00 000287D8 8F DD 0002B 3$:      PUSHL  #165848
          01 FB 00031      CALLS  #1, LIB$SIGNAL
          52 D0 00038 4$:      MOVL   LENGTH, R0
          04 0003B      RET

```

0594  
0596  
0597

; Routine Size: 60 bytes, Routine Base: DBG\$CODE + 0248

```

470      0598      2
471      0599
472      0600
473      0601      LOCAL
474      0602      SOURCE_NN      : REF DBG$NOUN_NODE,      ! Source of deposit
475      0603      TARGET_NN     : REF DBG$NOUN_NODE,      ! Target of deposit
476      0604      TYPE_NODE     : REF DBG$ADVERB_NODE,    ! Command qualifier
477      0605      PRIM_DESC     : REF DBG$PRIMARY,
478      0606      ADDR_DESC     : REF DBG$VALDESC,
479      0607      DATA_DESC    : REF DBG$VALDESC,
480      0608      MESSAGE_VECT;      ! Error message vector
481      0609
482      0610      BUILTIN CALLG;
483      0611      ENABLE DEPOSIT_HANDLER;
484      0612
485      0613      TARGET_NN = .VERB_NODE[DBG$L_VERB_OBJECT_PTR];
486      0614      SOURCE_NN = .TARGET_NN[DBG$NOUN_LINK];
487      0615      PRIM_DESC = .TARGET_NN[DBG$NOUN_VALUE];
488      0616      DATA_DESC = .SOURCE_NN[DBG$NOUN_VALUE];
489      0617      PAGE_LIST = 0;
490      0618
491      0619      ! Convert both the source and the target to value descriptors.
492      0620      ! eval_lang operator is used to convert the source because it
493      0621      ! is sensitive to any language-specific rules for converting
494      0622      ! primaries to values (e.g., in BLISS we do primary->address,
495      0623      ! in other languages we do primary->value).
496      0624
497      0625      IF .DATA_DESC[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
498      0626      THEN
499      0627          DATA_DESC = DBG$EVAL_LANG_OPERATOR(DBG$GL_IDENTITY_TOKEN,
500      0628              .DATA_DESC, 0);
501      0629      DBG$PRIM_TO_VAL(.PRIM_DESC, DBG$K_V_VALUE_DESC, ADDR_DESC);
502      0630
503      0631      IF (TYPE_NODE = .VERB_NODE[DBG$L_VERB_ADVERB_PTR]) EQLA 0
504      0632      THEN DBG$SAVE_LOC(.PRIM_DESC)
505      0633      ELSE
506      0634          BEGIN
507      0635              LOCAL OVERRIDE TYPE, OVERRIDE SIZE;
508      0636              ADDR_DESC[DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
509      0637              ADDR_DESC[DBG$V_DHDR_OVERRIDE] = TRUE;
510      0638              OVERRIDE_TYPE = .TYPE_NODE[DBG$B_ADVERB_LITERAL];
511      0639              OVERRIDE_SIZE = .TYPE_NODE[DBG$L_ADVERB_VALUE];
512      0640              SELECTONE .OVERRIDE_TYPE OF
513      0641                  SET
514      0642                  [DSC$K_DTYPE_AZ, DSC$K_DTYPE_AC, DSC$K_DTYPE_VT];
515      0643              BEGIN

```

```

516 0644 4 IF .ADDR_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_UBS THEN SIGNAL(DBG$_UNALIGNED);
517 0645 4 ADDR_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_VS;
518 0646 4 ADDR_DESC[DBG$B_VALUE_DTYPE] = .OVERRIDE_TYPE;
519 0647 4 ADDR_DESC[DBG$W_VALUE_LENGTH] = TEXT_LENGTH(.DATA_DESC);
520 0648 4 END;
521 0649 4
522 0650 4 [DBG$K_DTYPE_AD]:
523 0651 4 IF NOT CHECK_TEXT_DESCRIPTOR(.ADDR_DESC) THEN SIGNAL(DBG$_DESCNOTSET);
524 0652 4
525 0653 4 [OTHERWISE]:
526 0654 4 BEGIN
527 0655 4 IF (.ADDR_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS)
528 0656 4 THEN
529 0657 4 BEGIN
530 0658 4 IF (.OVERRIDE_SIZE GTR 32)
531 0659 4 OR (.OVERRIDE_TYPE EQL DSC$K_DTYPE_ZI)
532 0660 4 OR (.OVERRIDE_TYPE EQL DSC$K_DTYPE_T)
533 0661 4 THEN
534 0662 4 SIGNAL(DBG$_UNALIGNED);
535 0663 4 END
536 0664 4 ELSE
537 0665 4 ADDR_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_Z;
538 0666 4
539 0667 4 IF (.OVERRIDE_TYPE EQL DSC$K_DTYPE_T)
540 0668 4 AND
541 0669 4 (.OVERRIDE_SIZE EQL 0)
542 0670 4 THEN OVERRIDE_SIZE = TEXT_LENGTH(.DATA_DESC);
543 0671 4
544 0672 4 ADDR_DESC[DBG$B_VALUE_DTYPE] = .OVERRIDE_TYPE;
545 0673 4 IF .OVERRIDE_TYPE EQL DSC$K_DTYPE_ZI
546 0674 4 THEN
547 0675 4 BEGIN
548 0676 4 LOCAL
549 0677 4 ADDR;
550 0678 4
551 0679 4 ADDR = .ADDR_DESC[DBG$L_VALUE_POINTER];
552 0680 4 ADDR_DESC[DBG$W_VALUE_LENGTH] =
553 0681 4 DBG$INS_DECODE(.ADDR, FALSE, FALSE) - .ADDR;
554 0682 4 END
555 0683 4
556 0684 4 ELSE
557 0685 4 ADDR_DESC[DBG$W_VALUE_LENGTH] = .OVERRIDE_SIZE;
558 0686 4
559 0687 4 END;
560 0688 4
561 0689 4 TES:
562 0690 4 DBG$SAVE_LOC(.PRIM_DESC, ADDR_DESC[DBG$A_VALUE_VMSDESC]);
563 0691 4 END;
564 0692 4
565 0693 4 DBG$SAVE_VAL(.DATA_DESC);
566 0694 4
567 0695 4 IF NOT DBG$NGET_PAGES(.PRIM_DESC, PAGE_LIST, MESSAGE_VECT)
568 0696 4 OR NOT DBG$SET_PAGE_PROT(PAGE_LIST, FALSE, MESSAGE_VECT)
569 0697 4 THEN
570 0698 4 BEGIN
571 0699 4 PAGE_LIST = 0;
572 0700 4 CALLG(.MESSAGE_VECT, LIB$SIGNAL);

```

```

573 0701 2
574 0702 2
575 0703 2
576 0704 2
577 0705 2
578 0706 2
579 0707 2
580 0708 2
581 0709 2
582 0710 2
583 0711 2
584 0712 2
585 0713 2
586 0714 2
587 0715 2
588 0716 2
589 0717 2
590 0718 2
591 0719 2
592 0720 1

```

```

END;
DBG$EVAL_LANG_OPERATOR(DBG$GL_DEPOSIT_TOKEN,.DATA_DESC,.ADDR_DESC);
IF NOT DBG$SET_PAGE_PROT(PAGE_LIST,TRUE,MESSAGE_VECT)
THEN
  BEGIN
    PAGE_LIST = 0;
    CALLG(.MESSAGE_VECT,LIB$SIGNAL);
  END;
  ! Set registers context and return
  DBG$STA_SETREGISTERS();
  ! Update all watch point event entries after the DEPOSIT.
  DBG$UPDATE_WATCHPOINTS();
END;

```

			OFFC	00000	.ENTRY		
					DBG\$DEPOSIT, Save R2,R3,R4,R5,R6,R7,R8,R9,-		0559
					R10,R11		
					MOVAB DBG\$SAVE_LOC, R11		
					MOVAB DBG\$EVAL_LANG_OPERATOR, R10		
					MOVAB LIB\$SIGNAL, R9		
					MOVAB PAGE_LIST, R8		
					SUBL2 #8, SP		
					MOVAL 16\$, (FP)		0583
					MOVL VERB_NODE, R2		0613
					MOVL 8(R2), TARGET_NN		
					MOVL 8(TARGET_NN), SOURCE_NN		0614
					MOVL (TARGET_NN), PRIM_DESC		0615
					MOVL (SOURCE_NN), DATA_DESC		0616
					CLRL PAGE_LIST		0617
					CMPEB 2(DATA_DESC), #121		0625
					BNEQ 1\$		
					CLRL -(SP)		0627
					PUSHL DATA_DESC		0628
					PUSHAB DBG\$GL_IDENTITY_TOKEN		0627
					CALLS #3, DBG\$EVAL_LANG_OPERATOR		
					MOVL R0, DATA_DESC		
					PUSHL SP		0629
					MOVZBL #131, -(SP)		
					PUSHL PRIM_DESC		
					CALLS #3, DBG\$PRIM_TO_VAL		
					MOVL 4(R2), TYPE_NODE		0631
					BNEQ 2\$		
					PUSHL PRIM_DESC		0632
					CALLS #1, DBG\$SAVE_LOC		
					BRW 12\$		
					MOVL ADDR_DESC, R2		0636
					MOVB #3, 8(R2)		



04	A2	B0	8F	88	00075	BISB2	#128, 4(R2)	0637	
	54		63	9A	0007A	MOVZBL	(TYPE NODE), OVERRIDE_TYPE	0638	
	55	04	A3	D0	0007D	MOVL	4(TYPE NODE), OVERRIDE_SIZE	0639	
	25		54	D1	00081	CMPL	OVERRIDE_TYPE, #37	0642	
			29	19	00084	BLSS	4\$		
	27		54	D1	00086	CMPL	OVERRIDE_TYPE, #39		
			24	14	00089	BGTR	4\$		
	0D	0C	A2	91	0008B	CMPB	12(R2), #13	0644	
			09	12	0008F	BNEQ	3\$		
		00028D08	8F	DD	00091	PUSHL	#167176		
	69		01	FB	00097	CALLS	#1, LIB\$SIGNAL		
17	A2		0B	90	0009A	3\$:	MOVB	#11, 23(R2)	0645
16	A2		54	90	0009E	MOVB	OVERRIDE_TYPE, 22(R2)	0646	
			56	DD	000A2	PUSHL	DATA_DESC	0647	
FF1B	CF		01	FB	000A4	CALLS	#1, TEXT_LENGTH		
14	A2		50	B0	000A9	MOVW	R0, 20(R2)		
			75	11	000AD	BRB	11\$	0640	
	38		54	D1	000AF	4\$:	CMPL	OVERRIDE_TYPE, #56	0650
			15	12	000B2	BNEQ	5\$		
			52	DD	000B4	PUSHL	R2	0651	
0000V	CF		01	FB	000B6	CALLS	#1, CHECK_TEXT_DESCRIPTOR		
	66		50	E8	000BB	BLBS	R0, 11\$		
		00028F50	8F	DD	000BE	PUSHL	#167760		
	69		01	FB	000C4	CALLS	#1, LIB\$SIGNAL		
			5B	11	000C7	BRB	11\$		
	53	14	A2	9E	000C9	5\$:	MOVAB	20(R2), R3	0655
	0D	03	A3	91	000CD	CMPB	3(R3), #13		
			1A	12	000D1	BNEQ	7\$		
	20		55	D1	000D3	CMPL	OVERRIDE_SIZE, #32	0658	
			0A	14	000D6	BGTR	6\$		
	16		54	D1	000D8	CMPL	OVERRIDE_TYPE, #22	0659	
			05	13	000DB	BEQL	6\$		
	0E		54	D1	000DD	CMPL	OVERRIDE_TYPE, #14	0660	
			0E	12	000E0	BNEQ	8\$		
		00028D08	8F	DD	000E2	6\$:	PUSHL	#167176	0662
	69		01	FB	000E8	CALLS	#1, LIB\$SIGNAL		
			03	11	000EB	BRB	8\$	0655	
		03	A3	94	000ED	7\$:	CLRB	3(R3)	0665
	0E		54	D1	000F0	8\$:	CMPL	OVERRIDE_TYPE, #14	0667
			0E	12	000F3	BNEQ	9\$		
			55	D5	000F5	TSTL	OVERRIDE_SIZE	0669	
			0A	12	000F7	BNEQ	9\$		
			56	DD	000F9	PUSHL	DATA_DESC	0670	
FEC4	CF		01	FB	000FB	CALLS	#1, TEXT_LENGTH		
	55		50	D0	00100	MOVL	R0, OVERRIDE_SIZE		
02	A3		54	90	00103	9\$:	MOVB	OVERRIDE_TYPE, 2(R3)	0672
	16		54	D1	00107	CMPL	OVERRIDE_TYPE, #22	0673	
			15	12	0010A	BNEQ	10\$		
	54	18	A2	D0	0010C	MOVL	24(R2), ADDR	0679	
			7E	7C	00110	CLRQ	-(SP)	0681	
			54	DD	00112	PUSHL	ADDR		
00000000G	00		03	FB	00114	CALLS	#3, DBG\$INS DECODE		
63	50		54	A3	0011B	SUBW3	ADDR, R0, (R3)		
			03	11	0011F	BRB	11\$	0673	
	63		55	B0	00121	10\$:	MOVW	OVERRIDE_SIZE, (R3)	0685
		14	A2	9F	00124	11\$:	PUSHAB	20(R2)	0690
			57	DD	00127	PUSHL	PRIM_DESC		

6B	02	FB	00129	CALLS	#2, DBG\$SAVE_LOC	...
00000000G	00	56	DD 0012C	PUSHL	DATA_DESC	0693
	04	01	FB 0012E	CALLS	#1, DBG\$SAVE_VAL	...
	0180	AE	9F 00135	PUSHAB	MESSAGE_VECT	0695
00000000G	00	8F	BB 00138	PUSHR	#*M<R7,R8>	...
11	03	FB	0013C	CALLS	#3, DBG\$NGET_PAGES	...
	50	E9	00143	BLBC	R0, 13\$	...
	04	AE	9F 00146	PUSHAB	MESSAGE_VECT	0696
	7E	D4	00149	CLRL	-(SP)	...
00000000G	00	58	DD 0014B	PUSHL	R8	...
06	03	FB	0014D	CALLS	#3, DBG\$SET_PAGE_PROT	...
	50	E8	00154	BLBS	R0, 14\$	...
	68	D4	00157	CLRL	PAGE_LIST	0699
69	04	BE	FA 00159	CALLG	@MESSAGE_VECT, LIB\$SIGNAL	0700
	6E	DD	0015D	PUSHL	ADDR_DESC	0703
	56	DD	0015F	PUSHL	DATA_DESC	...
00000000G	00	9F	00161	PUSHAB	DBG\$GL_DEPOSIT_TOKEN	...
6A	03	FB	00167	CALLS	#3, DBG\$EVAL_LANG_OPERATOR	...
	04	AE	9F 0016A	PUSHAB	MESSAGE_VECT	0705
	01	DD	0016D	PUSHL	#1	...
00000000G	00	58	DD 0016F	PUSHL	R8	...
06	03	FB	00171	CALLS	#3, DBG\$SET_PAGE_PROT	...
	50	E8	00178	BLBS	R0, 15\$	...
	68	D4	0017B	CLRL	PAGE_LIST	0708
69	04	BE	FA 0017D	CALLG	@MESSAGE_VECT, LIB\$SIGNAL	0709
00000000G	00	00	FB 00181	CALLS	#0, DBG\$STA_SETREGISTERS	0714
00000000G	00	00	FB 00188	CALLS	#0, DBG\$UPDATE_WATCHPOINTS	0718
	04	0018F	RET			0720
	0000	00190	.WORD	Save nothing		0583
	7E	D4	00192	CLRL	-(SP)	...
	5E	DD	00194	PUSHL	SP	...
FDF4	7E	AC	7D 00196	MOVQ	4(AP), -(SP)	...
CF	03	FB	0019A	CALLS	#3, DEPOSIT_HANDLER	...
	04	0019F	RET			...

; Routine Size: 416 bytes, Routine Base: DBG\$CODE + 0284

```
594 0721 1 GLOBAL ROUTINE DBG$EVALUATE(VERB_NODE): NOVALUE =
595 0722 1
596 0723 1 FUNCTION
597 0724 1 This routine is the command execution network for the EVALUATE command.
598 0725 1 Various semantic actions are performed which correspond to the arguments
599 0726 1 and operands of the parsed input string.
600 0727 1
601 0728 1 EVALUATE sets last val '\', EVALUTATE/ADDRESS sets '.', current loc.
602 0729 1
603 0730 1 INPUTS
604 0731 1 VERB_NODE - A longword containing the address of the head
605 0732 1 node in the command execution tree
606 0733 1
607 0734 1 OUTPUTS
608 0735 1 NONE
609 0736 1
610 0737 1 BEGIN
611 0738 1
612 0739 1 MAP
613 0740 1 VERB_NODE: REF DBG$VERB_NODE; ! Pointer to the input Verb Node
614 0741 1
615 0742 1 LOCAL
616 0743 1 RADIX,
617 0744 1 NOUN_NODE: REF DBG$NOUN_NODE,
618 0745 1 BASE_NODE: REF DBG$ADVERB_NODE,
619 0746 1 PRM_DESC: REF DBG$PRIMARY,
620 0747 1 VAL_DESC: REF DBG$VALDESC;
621 0748 1
622 0749 1
623 0750 1
624 0751 1
625 0752 1
626 0753 1 ! Flush the current print buffer. Then pick up the first Noun Node pointer,
627 0754 1 the Adverb Node pointer, and the radix setting for this command.
628 0755 1
629 0756 1 DBG$FLUSHBUF();
630 0757 1 NOUN_NODE = .VERB_NODE [DBG$NOUN_OBJECT_PTR];
631 0758 1 BASE_NODE = .VERB_NODE [DBG$NOUN_ADVERB_PTR];
632 0759 1 IF .BASE_NODE EQL 0
633 0760 1 THEN
634 0761 1 RADIX = DBG$K_DEFAULT
635 0762 1 ELSE
636 0763 1 RADIX = .BASE_NODE[DBG$NOUN_ADVERB_LITERAL];
637 0764 1
638 0765 1
639 0766 1 ! Loop through all the Noun Nodes to process each expression on the
640 0767 1 EVALUATE command.
641 0768 1
642 0769 1 WHILE .NOUN_NODE NEQ 0 DO
643 0770 1 BEGIN
644 0771 1 PRM_DESC = .NOUN_NODE[DBG$NOUN_VALUE];
645 0772 1 DBG$COLLECT(.PRM_DESC);
646 0773 1
647 0774 1
648 0775 1 ! Case on the kind of EVALUATE command Verb Node we have as determined
649 0776 1 by the command qualifiers.
650 0777 1
```



```
CASE .VERB_NODE(DBG$B_VERB_COMPOSITE) FROM EVALUATE TO EVALUATE_COND OF
SET

! Handle the plain EVALUATE and the EVALUATE/CONDITION_VALUE com-
! mands.
[EVALUATE,
EVALUATE_COND]:
BEGIN
  IF .PRM_DESC(DBG$V_DHDR_AGGR) THEN SIGNAL(DBG$ NOVALUE);
  IF .VERB_NODE(DBG$B_VERB_COMPOSITE) EQL EVALUATE_COND
  THEN
    PRM_DESC(DBG$V_DHDR_FORMAT) = 1

  ELSE IF .RADIX NEQ DBG$K_DEFAULT
  THEN
    PRM_DESC(DBG$V_DHDR_FORMAT) = 0;

  DBG$PRINT_VALUE(.PRM_DESC,.RADIX,.DBG$GL_SIGN_FLAG);
END;

! Handle the EVALUATE/ADDRESS command.
[EVALUATE_ADDR]:
BEGIN
  LOCAL
    NAMEPTR,
    REGDESCR,
    VMS_DESC: DBG$STG_DESC;

  DBG$SAVE_LOC(.PRM_DESC);
  DBG$PRM_TO_VAL(.PRM_DESC,DBG$K_V_VALUE_DESC,VAL_DESC);

  ! Check whether the address is in the register save area.
  REGDESCR = DBG$STA_ADDRESS_TO_REGDESCR(VAL_DESC(DBG$L_VALUE_POINTER));
  IF .REGDESCR NEQ 0 THEN
    BEGIN
      NAMEPTR = DBG$STA_REGISTER_NAME(.REGDESCR);
      DBG$PRINT(UPLOT BYTE(%ASCII '!AC'), .NAMEPTR);
    END
  ELSE
    BEGIN
      VMS_DESC[DSC$B_CLASS] = DSC$K_CLASS_2;
      VMS_DESC[DSC$B_DTYPE] = DSC$K_DTYPE_LU;
      VMS_DESC[DSC$B_LENGTH] = 4;
      VMS_DESC[DSC$B_POINTER] = VAL_DESC(DBG$L_VALUE_POINTER);
      DBG$PRINT_VALUE_AS_INTEGER(VMS_DESC,.RADIX);
    END;

  ! If the address is a bit_field then also print the <p,s,e>.
  !
```

```
651 0778
652 0779
653 0780
654 0781
655 0782
656 0783
657 0784
658 0785
659 0786
660 0787
661 0788
662 0789
663 0790
664 0791
665 0792
666 0793
667 0794
668 0795
669 0796
670 0797
671 0798
672 0799
673 0800
674 0801
675 0802
676 0803
677 0804
678 0805
679 0806
680 0807
681 0808
682 0809
683 0810
684 0811
685 0812
686 0813
687 0814
688 0815
689 0816
690 0817
691 0818
692 0819
693 0820
694 0821
695 0822
696 0823
697 0824
698 0825
699 0826
700 0827
701 0828
702 0829
703 0830
704 0831
705 0832
706 0833
707 0834
```

```

708 0835 4
709 0836
710 0837
711 0838
712 0839
713 0840
714 0841
715 0842
716 0843
717 0844
718 0845
719 0846
720 0847
721 0848
722 0849
723 0850
724 0851
725 0852
726 0853
727 0854
728 0855
729 0856
730 0857
731 0858
732 0859
733 0860
734 0861

DBG$PRINT_FIELD_REF(.VAL_DESC,TRUE);
END;

! Any other kind of Verb Node should never occur. If it does, we
! signal an internal DEBUG coding error.
[INRANGE,OUTRANGE]:
$DBG_ERROR('DBGLEVEL3\EVALUATE');
TES;

! Close out the current print line, link to the next Noun Node on the
! Noun Node list, and loop.
DBG$NEWLINE();
NOUN_NODE = .NOUN_NODE[DBG$L_NOUN_LINK];
END; ! End of WHILE loop over expressions

! The EVALUATE command is processed. Now return.
RETURN;
END;

```

```

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
4C 41 56 45 5C 33 4C 45 56 45 4C 43 41 21 03 0000 P.AAA: .ASCII <3>\!AC\
47 42 44 12 00004 P.AAB: .ASCII <18>\DBGLEVEL3\<92>\EVALUATE\
45 54 41 55 00013

```

```

.PSECT DBG$CODE,NOWRT, SHR, PIC,0
00000000G 57 00000000G 00 00FC 00000 .ENTRY DBG$EVALUATE, Save R2,R3,R4,R5,R6,R7 0721
5E 10 C2 00009 MOVAB LIB$SIGNAL, R7
00 00 FB 0000C SUBL2 #16, SP
54 04 AC D0 00013 CALLS #0, DBG$FLUSHBUF 0756
56 08 A4 D0 00017 MOVL VERB_NODE, R4 0757
50 04 A4 D0 0001B MOVL 8(R4), NOUN_NODE
05 12 0001F MOVL 4(R4), BASE_NODE
55 01 D0 00021 BNEQ 1$ 0758
03 11 00024 MOVL #1, RADIX 0759
55 60 9A 00026 BRB 2$ 0761
56 D5 00029 MOVZBL (BASE_NODE), RADIX 0763
01 12 0002B TSTL NOUN_NODE 0769
04 0002D BNEQ 3$
52 66 D0 0002E RET
FDOF CF 52 DD 00031 MOVL (NOUN_NODE), PRM_DESC 0771
01 FB 00033 PUSHL PRM_DESC 0772
CALLS #1, DBG$COLLECT

```

	02 0019	01 004F	01 0019	A4 0019	8F 00038 0003D 48:	CASEB .WORD	1(R4), #1, #2 58-48,- 108-48,- 58-48	0778
							P.AAB	0843
							#1	
							#164706	
							#3, LIB\$SIGNAL	
							98	
							4(PRM_DESC), 68	0788
							#165880	
							#1, LIB\$SIGNAL	
							1(R4), #3	0789
							78	
							#1, #4, #4, 5(PRM_DESC)	0791
							88	
							RADIX, #1	0793
							88	
							#240, 5(PRM_DESC)	0795
							DBG\$GL_SIGN_FLAG	0797
							#M<R2,R5>	
							#3, DBG\$PRINT_VALUE	
							138	0778
							PRM_DESC	0810
							#1, -DBG\$SAVE_LOC	
							SP	0811
							#131, -(SP)	
							PRM_DESC	
							#3, DBG\$PRIM_TO_VAL	
							VAL_DESC, R3	0816
							24(R3)	
							#1, DBG\$STA_ADDRESS_TO_REGDESCR	
							REGDESCR	0817
							118	
							REGDESCR	0819
							#1, DBG\$STA_REGISTER_NAME	
							NAMEPTR	0820
							P.AAA	
							#2, DBG\$PRINT	
							128	0817
							#262148, VMS_DESC	0827
							24(R3), VMS_DESC+4	0828
							RADIX	0829
							VMS_DESC	
							#2, -DBG\$PRINT_VALUE_AS_INTEGER	
							#1	0835
							R3	
							#2, DBG\$PRINT_FIELD_REF	
							#0, DBG\$NEWLINE	0851
							8(NOUN_NODE), NOUN_NODE	0852
							28	0769
							RET	0861

; Routine Size: 258 bytes, Routine Base: DBG\$CODE + 0424



736 0862 GLOBAL ROUTINE DBGSEXAMINE(VERB\_NODE: REF DBG\$VERB\_NODE): NOVALUE =  
737 0863  
738 0864

FUNCTION

This routine performs the action associated with EXAMINE xxx.  
We always get three adverb nodes linked to the verb node. See the  
routine header for DBG\$NPARSE\_EXAMINE in DBGNEXMNE.B32 for details.

INPUTS

VERB\_NODE - A longword containing the address of the command  
execution tree verb (head) node.

OUTPUTS

NONE

BEGIN

LOCAL

NOUN\_NODE : REF DBG\$NOUN\_NODE,  
TYPE\_NODE : REF DBG\$ADVERB\_NODE,  
BASE\_NODE : REF DBG\$ADVERB\_NODE,  
MODE\_NODE : REF DBG\$ADVERB\_NODE,  
PRM\_DESC : REF DBG\$PRIMARY,  
END\_DESC : REF DBG\$PRIMARY,  
VAL\_DESC : REF DBG\$VALDESC,  
NEW\_SIZE : WORD,  
NEW\_TYPE : BYTE,  
RADIX : BYTE,  
FORMAT\_ONE : BYTE,  
FORMAT\_TWO : BYTE;

NOUN\_NODE = .VERB\_NODE[DBG\$L\_VERB\_OBJECT\_PTR];  
TYPE\_NODE = .VERB\_NODE[DBG\$L\_VERB\_ADVERB\_PTR];  
BASE\_NODE = .TYPE\_NODE[DBG\$L\_ADVERB\_LINK];  
MODE\_NODE = .BASE\_NODE[DBG\$L\_ADVERB\_LINK];

SELECTONE .VERB\_NODE[DBG\$B\_VERB\_COMPOSITE] OF  
SET

[EXAMINE]:

BEGIN  
NEW\_TYPE = .TYPE\_NODE[DBG\$B\_ADVERB\_LITERAL];  
NEW\_SIZE = .TYPE\_NODE[DBG\$L\_ADVERB\_VALUE];  
RADIX = .BASE\_NODE[DBG\$B\_ADVERB\_LITERAL];  
FORMAT\_ONE = 0;  
END;

[EXAMINE\_SOURCE]: 0;

[EXAMINE\_CONDITION\_VALUE]:

BEGIN  
NEW\_TYPE = DSC\$K\_DTYPE\_LU;  
NEW\_SIZE = 4;  
RADIX = DBG\$K\_DEFAULT;  
FORMAT\_ONE = 1;  
END;

[EXAMINE\_PSL]:

792

```

BEGIN
NEW_TYPE    = DSCSK_DTYPE_LU;
NEW_SIZE    = 4;
RADIX       = DBGSK_DEFAULT;
FORMAT_ONE  = 2;
END;

[EXAMINE PSW]:
BEGIN
NEW_TYPE    = DSCSK_DTYPE_WU;
NEW_SIZE    = 2;
RADIX       = DBGSK_DEFAULT;
FORMAT_ONE  = 3;
END;

! Any other kind of the Verb Node is invalid, so we signal an internal
! DEBUG coding error.
[OTHERWISE]:
$DBG_ERROR('DBGLEVEL3\EXAMINE');

TES;

! -----
DO BEGIN
DBG$FLUSHBUF();

IF .VERB_NODE[DBG$B_VERB_COMPOSITE] EQL EXAMINE_SOURCE
THEN
BEGIN
LOCAL
VAL_DESC      : REF DBG$VALDESC,
START_ADDRESS,
FINAL_ADDRESS;

DBG$PRIM TO VAL(.NOUN_NODE[DBG$N_NOUN_VALUE ],DBG$K_V_VALUE_DESC,VAL_DESC);
START_ADDRESS = .VAL_DESC[DBG$N_VALUE_POINTER];
DBG$PRIM TO VAL(.NOUN_NODE[DBG$N_NOUN_VALUE2],DBG$K_V_VALUE_DESC,VAL_DESC);
FINAL_ADDRESS = .VAL_DESC[DBG$N_VALUE_POINTER];

! Output the source. The third parameter indicates that the
! module name is to be displayed.

DBG$SRC_TYPE_PC_SOURCE(.START_ADDRESS,.FINAL_ADDRESS,TRUE,FALSE);

PRM_DESC = .NOUN_NODE[DBG$N_NOUN_VALUE2];
Commented out because screen window does EXAMINE/SOURCE and
we don't want to save dot there.

DBG$SAVE_LOC(.PRM_DESC);
END      ! EXAMINE/SOURCE

! -----

```

```

793 0919
794 0920
795 0921
796 0922
797 0923
798 0924
799 0925
800 0926
801 0927
802 0928
803 0929
804 0930
805 0931
806 0932
807 0933
808 0934
809 0935
810 0936
811 0937
812 0938
813 0939
814 0940
815 0941
816 0942
817 0943
818 0944
819 0945
820 0946
821 0947
822 0948
823 0949
824 0950
825 0951
826 0952
827 0953
828 0954
829 0955
830 0956
831 0957
832 0958
833 0959
834 0960
835 0961
836 0962
837 0963
838 0964
839 0965
840 0966
841 0967
842 0968
843 0969
844 0970
845 0971
846 0972
847 0973
848 0974
849 0975

```

```

850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906

```

```

ELSE
BEGIN
    ! Data Examine
    PRM_DESC = .NOUN_NODE(DBGSL_NOUN_VALUE);
    END_DESC = .NOUN_NODE(DBGSL_NOUN_VALUE2);
    DBG$COLLECT(.PRM_DESC);
    DBG$COLLECT(.END_DESC);

    IF (.END_DESC NEQ 0) AND (.PRM_DESC NEQ .END_DESC)
    THEN
        BEGIN
            +
            We have a ranged examine (EXAMINE <prm>:<end>)
            Check for the case where the two endpoints are part
            of th same structure. We have to ensure that a number
            of conditions are met, e.g., they are both primaries,
            they are not aggregates, and so on.
            -
            IF
                (.PRM_DESC[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC)
                AND
                (.END_DESC[DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC)
                AND
                (.PRM_DESC[DBG$B_DHDR_SYMID0] EQL .END_DESC[DBG$B_DHDR_SYMID0])
                AND
                (.NEW_TYPE EQL DBG$K_NOTYPE)
                AND
                (NOT .PRM_DESC[DBG$V_DHDR_AGG])
                AND
                (NOT .END_DESC[DBG$V_DHDR_AGG])
                AND
                (NOT .PRM_DESC[DBG$V_DHDR_SUBREF])
                AND
                (NOT .END_DESC[DBG$V_DHDR_SUBREF])
            THEN
                BEGIN
                    +
                    The start and end of the ranged examine appear to be
                    part of the same aggregate structure. Check that the
                    start is earlier than the end
                    -
                    IF PRIMARY_ORDER(.PRM_DESC,.END_DESC) GTR 0 THEN SIGNAL(DBG$EXARANGE);
                    WHILE TRUE DO
                        BEGIN
                            LOCAL MARK;
                            MARK = DBG$PUSH_TEMPMEM();
                            DBG$PRINT IDENTIFIER(.PRM_DESC);
                            DBG$PRINT(UPLIT BYTE(ASCII '!AD! '), 1, UPLIT BYTE(':'));
                            DBG$PRINT TO VAL(.PRM_DESC,DBG$K_VALUE_DESC,VAL_DESC);
                            IF .FORMAT ONE NEQ 0 THEN VAL_DESC[DBG$V_DHDR_FORMAT] = .FORMAT_ONE;
                            DBG$PRINT VALUE(.VAL_DESC,.RADIX,.DBG$G_SIGN_FLAG);
                            DBG$NEWLINE();
                            DBG$SAVE_LOC(.PRM_DESC);
                            DBG$POP_TEMPMEM(.MARK);
                            IF PRIMARY_ORDER(.PRM_DESC,.END_DESC) GEQ 0 THEN EXITLOOP;
                            IF NOT MODIFY_PRIMARY(.PRM_DESC,0) THEN EXITLOOP;
                        END;
                    END;
                END;
            END;
        END;
    END;

```



```

907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963

```

```

1033 5
1034 6
1035 6
1036 6
1037 6
1038 6
1039 6
1040 6
1041 6
1042 6
1043 6
1044 6
1045 6
1046 6
1047 6
1048 6
1049 6
1050 6
1051 6
1052 6
1053 6
1054 6
1055 6
1056 6
1057 6
1058 6
1059 6
1060 5
1061 7
1062 6
1063 6
1064 6
1065 6
1066 6
1067 7
1068 6
1069 6
1070 6
1071 6
1072 6
1073 6
1074 6
1075 7
1076 7
1077 7
1078 7
1079 7
1080 7
1081 7
1082 7
1083 7
1084 7
1085 7
1086 7
1087 7
1088 7
1089 7

```

```

ELSE
BEGIN
    ! The start and end are NOT part of the same aggregate.
    LOCAL
        MARK,
        LAST_ADDR,
        NEXT_ADDR,
        DESC_TYPE,
        ADDR_DESC      : REF DBG$VALDESC,
        RDESC_ONE      : DBG$REGDESCR,
        RDESC_TWO      : DBG$REGDESCR,
        LENGTH;

    ADDR_DESC = DBG$CHANGE_DTYPE(.END_DESC, .NEW_TYPE, .NEW_SIZE);
    RDESC_ONE = DBG$STA_ADDRESS_TO_REGDESCR(.ADDR_DESC[DBG$L_VALUE_POINTER]);
    LAST_ADDR = .ADDR_DESC[DBG$C_VALUE_POINTER];
    ADDR_DESC = DBG$CHANGE_DTYPE(.PRM_DESC, .NEW_TYPE, .NEW_SIZE);
    RDESC_TWO = DBG$STA_ADDRESS_TO_REGDESCR(.ADDR_DESC[DBG$L_VALUE_POINTER]);
    IF ((.RDESC_ONE XOR .RDESC_TWO) AND %X'FFFF00FC') NEQ 0
    THEN
        SIGNAL(DBG$_EXARANGE);

    IF .LAST_ADDR LSSA .ADDR_DESC[DBG$L_VALUE_POINTER]
    THEN
        SIGNAL(DBG$_EXARANGE);

    IF (.ADDR_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS)
    THEN
        SIGNAL(DBG$_ILLTYPE);

    DESC_TYPE = DBG$K_VALUE_DESC;
    IF (.ADDR_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZEM) OR
        (.ADDR_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZI)
    THEN
        DESC_TYPE = DBG$K_V_VALUE_DESC;

    ! -----
    WHILE TRUE DO
    BEGIN
        MARK = DBG$PUSH_TEMPMEM();
        DBG$PRINT_IDENTIFIER(.ADDR_DESC);
        DBG$PRINT(UPBIT BYTE(%ASCII 'AD! '), 1, UPBIT BYTE(':'));
        DBG$PRIM TO VAL(.ADDR_DESC, .DESC_TYPE, VAL_DESC);
        IF .FORMAT_ONE NEQ 0 THEN VAL_DESC[DBG$V_BHDR_FORMAT] = .FORMAT_ONE;
        DBG$PRINT_VALUE(.VAL_DESC, .RADIX, .DBG$G_SIGN_FLAG);
        DBG$NEWLINE();
        DBG$POP_TEMPMEM(.MARK);

        ! Get the increment we will add to the address for
        ! the next line of the ranged examine. If the increment
        ! is zero then signal an informational and get out of the loop.
        LENGTH = (DBG$DATA_LENGTH(ADDR_DESC[DBG$A_VALUE_VMSDESC]) + (%BPUNIT-1))/%BPUNIT;
    END

```

```

964 1090 7
965 1091 7
966 1092 8
967 1093 8
968 1094 8
969 1095 7
970 1096 7
971 1097 7
972 1098 7
973 1099 7
974 1100 9
975 1101 8
976 1102 7
977 1103 8
978 1104 8
979 1105 8
980 1106 8
981 1107 8
982 1108 8
983 1109 7
984 1110 7
985 1111 6
986 1112 6
987 1113 5
988 1114 5
989 1115 5
990 1116 5
991 1117 5
992 1118 5
993 1119 5
994 1120 5
995 1121 5
996 1122 5
997 1123 5
998 1124 5
999 1125 5
1000 1126 5
1001 1127 5
1002 1128 5
1003 1129 5
1004 1130 5
1005 1131 5
1006 1132 6
1007 1133 6
1008 1134 6
1009 1135 6
1010 1136 6
1011 1137 6
1012 1138 6
1013 1139 7
1014 1140 7
1015 1141 7
1016 1142 6
1017 1143 6
1018 1144 6
1019 1145 6
1020 1146 7

IF .LENGTH EQL 0
THEN
BEGIN
SIGNAL(DBG$ZEROINCR); ! Informational
EXITLOOP;
END;

NEXT ADDR = .ADDR_DESC[DBG$VALUE_POINTER] + .LENGTH;
IF .NEXT_ADDR GTR .LAST_ADDR THEN EXITLOOP;
ADDR_DESC[DBG$VALUE_POINTER] = .NEXT_ADDR;
IF (.ADDR_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZEM)
OR (.ADDR_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZI)
THEN
BEGIN
IF DBG$IS_IT_ENTRY(.NEXT_ADDR)
THEN ADDR_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZEM
ELSE ADDR_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;
ADDR_DESC[DBG$W_VALUE_LENGTH] = DBG$INS_DECODE(.NEXT_ADDR,FALSE) - .NEXT_ADDR;
END
ELSE
ADDR_DESC[DBG$W_VALUE_LENGTH] = FIX_UP_LENGTH(ADDR_DESC[DBG$A_VALUE_VMSDESC]);
END;
DBG$SAVE_LOC(.ADDR_DESC);
END;
ELSE
BEGIN

! In the case where prm_desc is a volatile value descriptor
! representing an absolute address, the print_identifier
! will attempt to symbolize this address to a primary. If
! it succeeds, it will return the newly-constructed primary..
! In all other cases, it just returns the descriptor we pass
! into it, unchanged.

PRM_DESC = DBG$PRINT_IDENTIFIER(.PRM_DESC);
DBG$SAVE_LOC(.PRM_DESC);
IF .NEW_TYPE EQL DBG$K_NOTYPE AND .PRM_DESC[DBG$V_DHDR_AGR]
THEN
DBG$PRINT_AGGREGATE(.PRM_DESC,.RADIX)
ELSE
BEGIN
DBG$PRINT(UPLIT BYTE(ASCII '!AD! '), 1, UPLIT BYTE(':'));
VAL_DESC = DBG$CHANGE_DTYPE(.PRM_DESC,.NEW_TYPE,.NEW_SIZE);
FORMAT_TWO = .FORMAT_ONE;
IF .NEW_TYPE NEQ DBG$K_NOTYPE
THEN
DBG$SAVE_LOC(.PRM_DESC,VAL_DESC[DBG$A_VALUE_VMSDESC])
ELSE IF (.FORMAT_ONE EQL 0) AND (.RADIX EQL DBG$K_DEFAULT)
AND (.VAL_DESC[DBG$B_VALUE_CLASS] NEQ DSC$K_CLASS_UBS)
AND (.VAL_DESC[DBG$C_VALUE_POINTER] EQLA DBG$REG_VALUES[16])
THEN FORMAT_TWO = 2;

IF .VAL_DESC[DBG$B_VALUE_DTYPE] NEQ DSC$K_DTYPE_ZI
THEN
BEGIN

```

```

        DBG$PRIM TO VAL(.VAL_DESC,DBG$K_VALUE_DESC,VAL_DESC);
        DBG$DO_MAPPING(.VAL_DESC);
        END;

        IF .FORMAT TWO NEQ 0 THEN VAL_DESC[DBG$V_DHDR_FORMAT] = .FORMAT_TWO;
        DBG$PRINT VALUE(.VAL_DESC..RADIX,.DBG$GL_SIGN_FLAG);
        DBG$NEWLINE();
        END;
    END;
END UNTIL (NOUN_NODE = .NOUN_NODE[DBG$L_NOUN_LINK]) EQL 0;

RETURN ST$K_SUCCESS;
END;
```

[illegible]



06	5B	91	00048	2\$:	CMPB	R11, #6	0918	
	0E	12	0004B		BNEQ	3\$		
59	04	90	0004D		MOVB	#4, NEW_TYPE	0920	
6E	04	80	00050		MOVW	#4, NEW_SIZE	0921	
5A	01	90	00053		MOVB	#1, RADIX	0922	
58	02	90	00056		MOVB	#2, FORMAT_ONE	0923	
	28	11	00059		BRB	5\$	0898	
07	5B	91	0005B	3\$:	CMPB	R11, #7	0926	
	0E	12	0005E		BNEQ	4\$		
59	03	90	00060		MOVB	#3, NEW_TYPE	0928	
6E	02	80	00063		MOVW	#2, NEW_SIZE	0929	
5A	01	90	00066		MOVB	#1, RADIX	0930	
58	03	90	00069		MOVB	#3, FORMAT_ONE	0931	
	15	11	0006C		BRB	5\$	0898	
	00000000	EF	9F	0006E	4\$:	PUSHAB	P.AAC	0939
		01	DD	00074		PUSHL	#1	
	00028362	8F	DD	00076		PUSHL	#164706	
00000000G	00	03	FB	0007C		CALLS	#3, LIB\$SIGNAL	
00000000G	00	00	FB	00083	5\$:	CALLS	#0, DBG\$FLUSHBUF	0947
	04	5B	91	0008A		CMPB	R11, #4	0949
		46	12	0008D		BNEQ	6\$	
	18	AE	9F	0008F		PUSHAB	VAL_DESC	0957
7E	83	8F	9A	00092		MOVZBL	#13T, -(SP)	
		65	DD	00096		PUSHL	(NOUN_NODE)	
00000000G	00	03	FB	00098		CALLS	#3, DBG\$PRIM_TO_VAL	
	50	18	AE	D0	0009F	MOVL	VAL_DESC, R0	0958
	52	18	A0	D0	000A3	MOVL	24(R0), START_ADDRESS	
		18	AE	9F	000A7	PUSHAB	VAL_DESC	0959
7E	83	8F	9A	000AA		MOVZBL	#13T, -(SP)	
	0C	A5	DD	000AE		PUSHL	12(NOUN_NODE)	
00000000G	00	03	FB	000B1		CALLS	#3, DBG\$PRIM_TO_VAL	
	50	18	AE	D0	000B8	MOVL	VAL_DESC, R0	0960
	50	18	A0	D0	000BC	MOVL	24(R0), FINAL_ADDRESS	
7E		01	7D	000C0		MOVQ	#1, -(SP)	0965
		50	DD	000C3		PUSHL	FINAL_ADDRESS	
		52	DD	000C5		PUSHL	START_ADDRESS	
00000000G	00	04	FB	000C7		CALLS	#4, DBG\$SRC TYPE PC SOURCE	
	53	0C	A5	D0	000CE	MOVL	12(NOUN_NODE), PRM_DESC	0967
		03	5D	31	000D2	BRW	36\$	0949
	53		65	D0	000D5	MOVL	(NOUN_NODE), PRM_DESC	0979
	56	0C	A5	D0	000D8	MOVL	12(NOUN_NODE), END_DESC	0980
			53	DD	000DC	PUSHL	PRM_DESC	0981
FB62	CF		01	FB	000DE	CALLS	#1, DBG\$COLLECT	
			56	DD	000E3	PUSHL	END_DESC	0982
FB5B	CF		01	FB	000E5	CALLS	#1, DBG\$COLLECT	
			56	D5	000EA	TSTL	END_DESC	0984
			03	13	000EC	BEQL	7\$	
	56		53	D1	000EE	CMPL	PRM_DESC, END_DESC	
			03	12	000F1	BNEQ	8\$	
		02	58	31	000F3	BRW	29\$	
79	8F	02	A3	91	000F6	CMPB	2(PRM_DESC), #121	0994
			18	12	000FB	BNEQ	9\$	
79	8F	02	A6	91	000FD	CMPB	2(END_DESC), #121	0996
			11	12	00102	BNEQ	9\$	
0C	A6	0C	A3	D1	00104	CMPL	12(PRM_DESC), 12(END_DESC)	0998
			0A	12	00109	BNEQ	9\$	
80	8F		59	91	0010B	CMPB	NEW_TYPE, #128	1000

			03	04	04	12	0010F	BNEQ	9%		
					A3	E9	00111	BLBC	4(PRM_DESC), 10%		1002
					00B9	31	00115	BRW	15%		
					A6	E8	00118	BLBS	4(END_DESC), 9%		1004
F4	04	F9		04	01	E0	0011C	BBS	#1, 4(PRM_DESC), 9%		1006
EF	04	A3			01	E0	00121	BBS	#1, 4(END_DESC), 9%		1008
		A6			8F	BB	00126	PUSHR	#M<R3,R6>		1016
	0000V	CF		0048	02	FB	0012A	CALLS	#2, PRIMARY_ORDER		
					50	D5	0012F	TSTL	R0		
					0D	15	00131	BLEQ	11%		
				00028190	8F	DD	00133	PUSHL	#164240		
00000000G	00				01	FB	00139	CALLS	#1, LIBSSIGNAL		
00000000G	00				00	FB	00140	CALLS	#0, DBG\$PUSH_TEMPMEM		1020
	52				50	DD	00147	MOVL	R0, MARK		
					53	DD	0014A	PUSHL	PRM_DESC		1021
00000000G	00				01	FB	0014C	CALLS	#1, DBG\$PRINT_IDENTIFIER		
				00000000'	EF	9F	00153	PUSHAB	P.AAE		1022
					01	DD	00159	PUSHL	#1		
				00000000'	EF	9F	0015B	PUSHAB	P.AAD		
00000000G	00				03	FB	00161	CALLS	#3, DBG\$PRINT		
				1C	AE	9F	00168	PUSHAB	VAL_DESC		1023
	7E			7A	8F	9A	0016B	MOVZBL	#122, -(SP)		
					53	DD	0016F	PUSHL	PRM_DESC		
00000000G	00				03	FB	00171	CALLS	#3, DBG\$PRIM_TO_VAL		
					58	95	00178	TSTB	FORMAT_ONE		1024
					0A	13	0017A	BEQL	12%		
05	AD				AE	DD	0017C	MOVL	VAL_DESC, R0		
				04	5B	FD	00180	INSV	FORMAT_ONE, #4, #4, 5(R0)		
					00	DD	00186	PUSHL	DBG\$GL_SIGN_FLAG		1025
				7E	5A	9A	0018C	MOVZBL	RADIX, -(SP)		
					AE	DD	0018F	PUSHL	VAL_DESC		
00000000G	00			24	03	FB	00192	CALLS	#3, DBG\$PRINT_VALUE		
00000000G	00				00	FB	00199	CALLS	#0, DBG\$NEWLINE		1026
					53	DD	001A0	PUSHL	PRM_DESC		1027
00000000G	00				01	FB	001A2	CALLS	#1, DBG\$SAVE_LOC		
					52	DD	001A9	PUSHL	MARK		1028
00000000G	00				01	FB	001AB	CALLS	#1, DBG\$POP_TEMPMEM		
				0048	8F	BB	001B2	PUSHR	#M<R3,R6>		1029
0000V	CF				02	FB	001B6	CALLS	#2, PRIMARY_ORDER		
					50	D5	001BB	TSTL	R0		
					03	19	001BD	BLSS	14%		
					0270	31	001BF	BRW	36%		
					7E	D4	001C2	CLRL	-(SP)		1030
					53	DD	001C4	PUSHL	PRM_DESC		
0000V	CF				02	FB	001C6	CALLS	#2, MODIFY_PRIMARY		
	F1				50	E9	001CB	BLBC	R0, 13%		
					FF6F	31	001CE	BRW	11%		
					6E	3C	001D1	MOVZWL	NEW_SIZE, -(SP)		1048
	7E				59	9A	001D4	MOVZBL	NEW_TYPE, -(SP)		
	7E				56	DD	001D7	PUSHL	END_DESC		
F8FC	CF				03	FB	001D9	CALLS	#3, DBG\$CHANGE_DTYPE		
	52				50	DD	001DE	MOVL	R0, ADDR_DESC		
				18	A2	DD	001E1	PUSHL	24(ADDR_DESC)		1049
00000000G	00				01	FB	001E4	CALLS	#1, DBG\$STA_ADDRESS_TO_REGDESCR		
	54				50	DD	001EB	MOVL	R0, RDESC ONE		
0C	AE			18	A2	DD	001EE	MOVL	24(ADDR_DESC), LAST_ADDR		1050
	7E				6E	3C	001F3	MOVZWL	NEW_SIZE, -(SP)		1051

		7E		59	9A	001F6	MOVZBL	NEW_TYPE, -(SP)		
				53	DD	001F9	PUSHL	PRM_DESC		
	F8DA	CF		03	FB	001FB	CALLS	#3, DBG\$CHANGE_DTYPE		
		52		50	DD	00200	MOVL	R0, ADDR_DESC		
			18	A2	DD	00203	PUSHL	24(ADDR_DESC)	1052	
	00000000G	00		01	FB	00206	CALLS	#1, DBG\$STA_ADDRESS_TO_REGDESCR		
		50		54	CC	0020D	XORL2	RDESC_ONE, R0	1053	
	FFFF00FC	8F		50	D3	00210	BITL	R0, #-65284		
				0D	13	00217	BEQL	16\$		
			00028190	8F	DD	00219	PUSHL	#164240	1055	
	00000000G	00		01	FB	0021F	CALLS	#1, LIB\$SIGNAL		
		18	0C	AE	D1	00226	16\$:	CMPL	LAST_ADDR, 24(ADDR_DESC)	1057
				0D	1E	0022B	BGEQU	17\$		
			00028190	8F	DD	0022D	PUSHL	#164240	1059	
	00000000G	00		01	FB	00233	CALLS	#1, LIB\$SIGNAL		
		54	14	A2	9E	0023A	17\$:	MOVAB	20(ADDR_DESC), R4	1061
		0D	03	A4	91	0023E	CMPB	3(R4), #13		
				0D	12	00242	BNEQ	18\$		
			000287D8	8F	DD	00244	PUSHL	#165848	1063	
	00000000G	00		01	FB	0024A	CALLS	#1, LIB\$SIGNAL		
		08	7A	8F	9A	00251	18\$:	MOVZBL	#122, DESC_TYPE	1065
			02	A4	91	00256	CMPB	2(R4), #23	1066	
				06	13	0025A	BEQL	19\$		
		16	02	A4	91	0025C	CMPB	2(R4), #22	1067	
				05	12	00260	BNEQ	20\$		
	08	AE	83	8F	9A	00262	19\$:	MOVZBL	#131, DESC_TYPE	1069
	00000000G	00		00	FB	00267	20\$:	CALLS	#0, DBG\$PUSH_TEMPMEM	1076
		14		50	DD	0026E	MOVL	R0, MARK		
				52	DD	00272	PUSHL	ADDR_DESC	1077	
	00000000G	00		01	FB	00274	CALLS	#1, DBG\$PRINT_IDENTIFIER		
			00000000'	EF	9F	0027B	PUSHAB	P.AAG	1078	
				01	DD	00281	PUSHL	#1		
			00000000'	EF	9F	00283	PUSHAB	P.AAF		
	00000000G	00		03	FB	00289	CALLS	#3, DBG\$PRINT		
			1C	AE	9F	00290	PUSHAB	VAL_DESC	1079	
			0C	AE	DD	00293	PUSHL	DESC_TYPE		
				52	DD	00296	PUSHL	ADDR_DESC		
	00000000G	00		03	FB	00298	CALLS	#3, DBG\$PRIM_TO_VAL		
				58	95	0029F	TSTB	FORMAT_ONE	1080	
				0A	13	002A1	BEQL	21\$		
05	A0			AE	DD	002A3	MOVL	VAL_DESC, R0		
		04		58	FD	002A7	INSV	FORMAT_ONE, #4, #4, 5(R0)	1081	
			00000000G	00	DD	002AD	PUSHL	DBG\$GL_SIGN_FLAG		
		7E		5A	9A	002B3	MOVZBL	RADIX, -(SP)		
			24	AE	DD	002B6	PUSHL	VAL_DESC		
	00000000G	00		03	FB	002B9	CALLS	#3, DBG\$PRINT_VALUE	1082	
	00000000G	00		00	FB	002C0	CALLS	#0, DBG\$NEWLINE	1083	
			14	AE	DD	002C7	PUSHL	MARK		
	00000000G	00		01	FB	002CA	CALLS	#1, DBG\$POP_TEMPMEM	1089	
				54	DD	002D1	PUSHL	R4		
	00000000G	00		01	FB	002D3	CALLS	#1, DBG\$DATA_LENGTH		
		50		07	CD	002DA	ADDL2	#7, R0		
04	AE			08	C7	002DD	DIVL3	#8, R0, LENGTH		
		50		0F	12	002E2	BNEQ	22\$	1090	
			000287B3	8F	DD	002E4	PUSHL	#165811	1093	
	00000000G	00		01	FB	002EA	CALLS	#1, LIB\$SIGNAL		
				50	11	002F1	BRB	28\$	1092	

57	18	A2	04	AE	C1	002F3	228:	ADDL3	LENGTH, 24(ADDR_DESC), NEXT_ADDR	1097
	0C	AE		57	D1	002F9		CMPL	NEXT_ADDR, LAST_ADDR	1098
				44	1A	002FD		BGTRU	288	
	18	A2		57	D0	002FF		MOVL	NEXT_ADDR, 24(ADDR_DESC)	1099
		17	02	A4	91	00303		CMPB	2(R4), #23	1100
				06	13	00307		BEQL	238	
		16	02	A4	91	00309		CMPB	2(R4), #22	1101
				27	12	0030D		BNEQ	268	
				57	DD	0030F	238:	PUSHL	NEXT_ADDR	1104
	00000000G	00		01	FB	00311		CALLS	#1, DBG\$IS_IT_ENTRY	
		06		50	E9	00318		BLBC	R0, 248	
	02	A4		17	90	0031B		MOVB	#23, 2(R4)	1105
				04	11	0031F		BRB	258	
	02	A4		16	90	00321	248:	MOVB	#22, 2(R4)	1106
				7E	D4	00325	258:	CLRL	-(SP)	1107
				57	DD	00327		PUSHL	NEXT_ADDR	
	00000000G	00		02	FB	00329		CALLS	#2, DBG\$INS_DECODE	
64		50		57	A3	00330		SUBW3	NEXT_ADDR, R0, (R4)	
				0A	11	00334		BRB	278	1100
				54	DD	00336	268:	PUSHL	R4	1110
	0000V	CF		01	FB	00338		CALLS	#1, FIX_UP_LENGTH	
		64		50	B0	0033D		MOVW	R0, (R4)	
			FF24	31	00340	278:	BRW	208		1074
				52	DD	00343	288:	PUSHL	ADDR_DESC	1112
	00000000G	00		01	FB	00345		CALLS	#1, DBG\$SAVE_LOC	
				2B	11	0034C		BRB	308	0984
				53	DD	0034E	298:	PUSHL	PRM_DESC	1126
	00000000G	00		01	FB	00350		CALLS	#1, DBG\$PRINT_IDENTIFIER	
		53		50	D0	00357		MOVL	R0, PRM_DESC	
				53	DD	0035A		PUSHL	PRM_DESC	1127
	00000000G	00		01	FB	0035C		CALLS	#1, DBG\$SAVE_LOC	
	80	8F		59	91	00363		CMPB	NEW_TYPE, #128	1128
				13	12	00367		BNEQ	318	
		0F	04	A3	E9	00369		BLBC	4(PRM_DESC), 318	
		7E		5A	9A	0036D		MOVZBL	RADIX, -(SP)	1130
				53	DD	00370		PUSHL	PRM_DESC	
	00000000G	00		02	FB	00372		CALLS	#2, DBG\$PRINT_AGGREGATE	
			00B6	31	00379	308:	BRW	368		
			00000000'	EF	9F	0037C	318:	PUSHAB	P,AAI	1133
				01	DD	00382		PUSHL	#1	
			00000000'	EF	9F	00384		PUSHAB	P,AAH	
	00000000G	00		03	FB	0038A		CALLS	#3, DBG\$PRINT	
		7E		6E	3C	00391		MOVZWL	NEW_SIZE, -(SP)	1134
		7E		59	9A	00394		MOVZBL	NEW_TYPE, -(SP)	
				53	DD	00397		PUSHL	PRM_DESC	
	F73C	CF		03	FB	00399		CALLS	#3, DBG\$CHANGE_DTYPE	
	1C	AE		50	D0	0039E		MOVL	R0, VAL_DESC	
	10	AE		58	90	003A2		MOVB	FORMAT_ONE, FORMAT_TWO	1135
	80	8F		59	91	003A6		CMPB	NEW_TYPE, #128	1136
				10	13	003AA		BEQL	328	
7E	1C	AE		14	C1	003AC		ADDL3	#20, VAL_DESC, -(SP)	1138
				53	DD	003B1		PUSHL	PRM_DESC	
	00000000G	00		02	FB	003B3		CALLS	#2, DBG\$SAVE_LOC	
				28	11	003BA		BRB	338	
				58	95	003BC	328:	TSTB	FORMAT_ONE	1139
				24	12	003BE		BNEQ	338	
		01		5A	91	003C0		CMPB	RADIX, #1	



			50	1C	1F	12	003C3	BNEQ	338		
			0D	17	AE	D0	003C5	MOVL	VAL_DESC, R0		1140
					AO	91	003C9	CMPB	23(R0), #13		
			50	1C	15	13	003CD	BEQL	338		
			51	00000000G	AE	D0	003CF	MOVL	VAL_DESC, R0		1141
			51	18	00	9E	003D3	MOVAB	DBG\$REG_VALUES+64, R1		
					AO	D1	003DA	CMPL	24(R0), -R1		
					04	12	003DE	BNEQ	338		
10			AE		02	90	003E0	MOVB	#2, FORMAT_TWO		1142
			50	1C	AE	D0	003E4	MOVL	VAL_DESC, R0		1144
			16	16	AO	91	003E8	CMPB	22(R0), #22		
					1A	13	003EC	BEQL	348		
					1C	AE	9F	003EE	PUSHAB	VAL_DESC	1147
			7E	7A	8F	9A	003F1	MOVZBL	#122, -(SP)		
					50	DD	003F5	PUSHL	R0		
		00000000G	00		03	FB	003F7	CALLS	#3, DBG\$PRIM_TO_VAL		
		00000000G	00	1C	AE	DD	003FE	PUSHL	VAL_DESC		1148
					01	FB	00401	CALLS	#1, DBG\$DO_MAPPING		
				10	AE	95	00408	TSTB	FORMAT_TWO		1151
					0B	13	0040B	BEQL	358		
05	AO		50	1C	AE	D0	0040D	MOVL	VAL_DESC, R0		
		04	04	10	AE	F0	00411	INSV	FORMAT_TWO, #4, #4, 5(R0)		
					00	DD	00418	PUSHL	DBG\$GL_SIGN_FLAG		1152
			7E		5A	9A	0041E	MOVZBL	RADIX, -(SP)		
				24	AE	DD	00421	PUSHL	VAL_DESC		
		00000000G	00		03	FB	00424	CALLS	#3, DBG\$PRINT_VALUE		
		00000000G	00		00	FB	0042B	CALLS	#0, DBG\$NEWLINE		1153
			55	0B	AS	D0	00432	MOVL	8(NOUN_NODE), NOUN_NODE		1157
					03	13	00436	BEQL	378		
					FC4B	31	00438	BRW	58		
					04	0043B	378:	RET			1160

; Routine Size: 1084 bytes, Routine Base: DBG\$CODE + 0526

```

1036 1161 GLOBAL ROUTINE DBGSNEXTLOC(PRM_DESC) =
1037 1162
1038 1163 FUNCTION
1039 1164 -----
1040 1165
1041 1166 INPUTS
1042 1167 -----
1043 1168
1044 1169 OUTPUTS
1045 1170 -----
1046 1171
1047 1172
1048 1173 BEGIN
1049 1174
1050 1175 MAP
1051 1176     PRM_DESC: REF DBG$PRIMARY;      ! Pointer to Primary Descriptor
1052 1177
1053 1178 LOCAL
1054 1179     BYTE_OFFSET,
1055 1180     LENGTH,
1056 1181     REG_DESC: DBG$REGDESCR,
1057 1182     STATUS,
1058 1183     VAL_DESC: REF DBG$VALDESC;
1059 1184
1060 1185
1061 1186
1062 1187
1063 1188
1064 1189 STATUS = MODIFY_PRIMARY(.PRM_DESC,0);
1065 1190 IF .STATUS THEN RETURN .PRM_DESC;
1066 1191 IF .DBG$GL_CURLOC_VMSDESC NEQ 0
1067 1192 THEN
1068 1193     BEGIN
1069 1194         VAL_DESC = DBG$MAKE_VAL_DESC(.DBG$GL_CURLOC_VMSDESC, DBG$K_V_VALUE_DESC);
1070 1195         VAL_DESC[DBG$B_DHDR_LANG] = .PRM_DESC[DBG$B_DHDR_LANG];
1071 1196         VAL_DESC[DBG$L_DHDR_SYMD0] = .PRM_DESC[DBG$L_DHDR_SYMD0];
1072 1197     END
1073 1198
1074 1199 ELSE
1075 1200     BEGIN
1076 1201         IF .STATUS EQL 2 THEN SIGNAL(DBG$NOSUCC);
1077 1202         DBG$PRM_TO_VAL(.PRM_DESC, DBG$K_V_VALUE_DESC, VAL_DESC);
1078 1203     END;
1079 1204
1080 1205 IF (.VAL_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS)
1081 1206 THEN
1082 1207     SIGNAL(DBG$NOSUCC);
1083 1208
1084 1209 LENGTH = (DBG$DATA_LENGTH(VAL_DESC[DBG$A_VALUE_VMSDESC]) - 1)/%BPUNIT + 1;
1085 1210 REG_DESC = DBG$STA_ADDRESS_TO_REGDESCR(.VAL_DESC[DBG$L_VALUE_POINTER]);
1086 1211 IF .REG_DESC NEQ 0
1087 1212 THEN
1088 1213     BEGIN
1089 1214         BYTE_OFFSET = 4*.REG_DESC[DBG$B_REGD_REGNUM]
1090 1215                     + .REG_DESC[DBG$V_REGD_OFFSET]
1091 1216                     + .LENGTH * .DBG$GW_DF[TLENG];
1092 1217         IF (.BYTE_OFFSET GTR 16*%UPVAL) AND

```

```

1093 1218
1094 1219
1095 1220
1096 1221
1097 1222
1098 1223
1099 1224
1100 1225
1101 1226
1102 1227
1103 1228
1104 1229
1105 1230
1106 1231
1107 1232
1108 1233
1109 1234
1110 1235
1111 1236
1112 1237
1113 1238
1114 1239
1115 1240
1116 1241
1117 1242
1118 1243
1119 1244
1120 1245
1121 1246
1122 1247
1123 1248
1124 1249
1125 1250
1126 1251
1127 1252
1128 1253
1129 1254
1130 1255
1131 1256
1132 1257
1133 1258
1134 1259
1135 1260
1136 1261
1137 1262
1138 1263
1139 1264
1140 1265
1141 1266
1142 1267
1143 1268
1144 1269
1145 1270
1146 1271
1147 1272

```

```

((.DBG$GW_DFLTLENG NEQ 2) OR (.DBG$GW_DFLTLENG NEQ 4) OR
(.BYTE_OFFSET NEQ (16*%UPVAL + .DBG$GW_DFLTLENG)))
THEN
    SIGNAL(DBG$_NOSUCC);
END;

! Initialize the Value Descriptor to VMS descriptor class 2 (unknown) and
! set the pointer to the next location to be the current location plus the
! length of the current object.
VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_2;
VAL_DESC[DBG$L_VALUE_POINTER] = .VAL_DESC[DBG$L_VALUE_POINTER] + .LENGTH;

! If the data type is instruction or entry point, determine the type of the
! next location by seeing if it is an instruction or entry mask. Also com-
! pute its length by interpreting the instruction at that location.
IF (.VAL_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZI) OR
(.VAL_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZEM)
THEN
    BEGIN
        IF DBG$IS_IT_ENTRY(.VAL_DESC[DBG$L_VALUE_POINTER])
        THEN
            VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZEM;
        ELSE
            VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;
        VAL_DESC[DBG$W_VALUE_LENGTH] =
            DBG$INS_DECODE(.VAL_DESC[DBG$L_VALUE_POINTER], FALSE) -
            .VAL_DESC[DBG$L_VALUE_POINTER];
    END

! The next location is not an instruction or entry mask. It is thus a
! data object and we set up the Value Descriptor accordingly.
ELSE
    BEGIN
        VAL_DESC[DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
        VAL_DESC[DBG$V_DHDR_FORMAT] = 0;
        VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
        VAL_DESC[DBG$B_VALUE_DTYPE] = .DBG$GC_DFLTTP;
        VAL_DESC[DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
    END;

! Return a pointer to the Value Descriptor for the next location.
RETURN .VAL_DESC;
END;

```

			007C	00000	.ENTRY	DBG\$NEXTLOC, Save R2,R3,R4,R5,R6	1161
56	00000000G	00	9E	00002	MOVAB	DBG\$GW_DFLTLENG, R6	
55	00000000G	00	9E	00009	MOVAB	LIB\$SIGNAL, R5	
5E		04	C2	00010	SUBL2	#4, SP	
		7E	D4	00013	CLRL	-(SP)	1189
52	04	AC	D0	00015	MOVL	PRM_DESC, R2	
		52	DD	00019	PUSHL	R2	
0000V	CF	02	FB	0001B	CALLS	#2, MODIFY_PRIMARY	
03		50	E9	00020	BLBC	STATUS, 1\$	1190
		010A	31	00023	BRW	12\$	
51	00000000G	00	D0	00026	MOVL	DBG\$GL_CURLOC_VMSDESC, R1	1191
		1C	13	0002D	BEQL	2\$	
7E	83	8F	9A	0002F	MOVZBL	#131, -(SP)	1194
		51	DD	00033	PUSHL	R1	
00000000G	00	02	FB	00035	CALLS	#2, DBG\$MAKE_VAL_DESC	
	6E	50	D0	0003C	MOVL	R0, VAL_DESC	
03	A0	03	A2	90	MOVB	3(R2), 3(R0)	1195
0C	A0	0C	A2	D0	MOVL	12(R2), 12(R0)	1196
		1D	11	00049	BRB	4\$	1191
02		50	D1	0004B	CMPL	STATUS, #2	1201
		09	12	0004E	BNEQ	3\$	
	00028818	8F	DD	00050	PUSHL	#165912	
65		01	FB	00056	CALLS	#1, LIB\$SIGNAL	
		5E	DD	00059	PUSHL	SP	1202
7E	83	8F	9A	0005B	MOVZBL	#131, -(SP)	
		52	DD	0005F	PUSHL	R2	
00000000G	00	03	FB	00061	CALLS	#3, DBG\$PRIM_TO_VAL	
	52	6E	D0	00068	MOVL	VAL_DESC, R2	1205
	53	14	A2	9E	MOVAB	20(R2), R3	
0D	03	A3	91	0006F	CMPB	3(R3), #13	
		09	12	00073	BNEQ	5\$	
	00028818	8F	DD	00075	PUSHL	#165912	1207
65		01	FB	0007B	CALLS	#1, LIB\$SIGNAL	
		53	DD	0007E	PUSHL	R3	1209
00000000G	00	01	FB	00080	CALLS	#1, DBG\$DATA_LENGTH	
		50	D7	00087	DECL	R0	
		08	C6	00089	DIVL2	#8, R0	
54	01	A0	9E	0008C	MOVAB	1(R0), LENGTH	
	18	A2	DD	00090	PUSHL	24(R2)	1210
00000000G	00	01	FB	00093	CALLS	#1, DBG\$STA_ADDRESS_TO_REGDESCR	
		50	D5	0009A	TSTL	REG_DESC	1211
		3D	13	0009C	BEQL	7\$	
51	50	08	08	EF	EXTZV	#8, #8, REG_DESC, R1	1214
50	50	02	00	EF	EXTZV	#0, #2, REG_DESC, R0	1215
		50	6041	DE	MOVAL	(R0)(R1), R0	
51		50	54	C1	ADDL3	LENGTH, R0, R1	1216
		50	66	3C	MOVZWL	DBG\$GW_DFLTLENG, R0	
51		50	C0	000B3	ADDL2	R0, BYTE_OFFSET	
000000040	8F	51	D1	000B6	CMPL	BYTE_OFFSET, #64	1217
		1C	15	000BD	BLEQ	7\$	
02		50	B1	000BF	CMPL	R0, #2	1218
		0E	12	000C2	BNEQ	6\$	
04		50	B1	000C4	CMPL	R0, #4	
		09	12	000C7	BNEQ	6\$	



	50	40	A0	9E	000C9	MOVAB	64(R0), R0	1219
	50		51	D1	000CD	CMPL	BYTE_OFFSET, R0	
		00028B18	09	13	000D0	BEQL	78	
	65		8F	DD	000D2	68:	PUSHL	#165912
		03	01	FB	000D8	CALLS	#1, LIB\$SIGNAL	1221
18	A2		A3	94	000DB	78:	CLRB	3(R3)
	16	02	54	C0	000DE	ADDL2	LENGTH, 24(R2)	1230
			A3	91	000E2	CMPB	2(R3), #22	1231
	17	02	06	13	000E6	BEQL	88	1238
			A3	91	000E8	CMPB	2(R3), #23	1239
		18	2A	12	000EC	BNEQ	118	
00000000G	00		A2	DD	000EE	88:	PUSHL	24(R2)
	06		01	FB	000F1	CALLS	#1, DBG\$IS_IT_ENTRY	1242
02	A3		50	E9	000F8	BLBC	R0, 98	
			17	90	000FB	MOVB	#23, 2(R3)	1244
02	A3		04	11	000FF	BRB	108	
			16	90	00101	98:	MOVB	#22, 2(R3)
		18	7E	D4	00105	108:	CLRL	-(SP)
			A2	DD	00107	PUSHL	24(R2)	1250
00000000G	00		02	FB	0010A	CALLS	#2, DBG\$INS_DECODE	
63	50	18	A2	A3	00111	SUBW3	24(R2), R0, (R3)	1251
			18	11	00116	BRB	128	1238
06	A2		03	90	00118	118:	MOVB	#3, 6(R2)
05	A2	F0	8F	8A	0011C	BICB2	#240, 5(R2)	1260
03	A3		01	90	00121	MOVB	#1, 3(R3)	1261
02	A3	00000000G	00	90	00125	MOVB	DBG\$GL_DFLTTP, 2(R3)	1262
	63		66	B0	0012D	MOVW	DBG\$GW_DFLTLENG, (R3)	1263
	50		52	D0	00130	128:	MOVL	R2, R0
			04	00133	RET			1270
								1272

; Routine Size: 308 bytes, Routine Base: DBG\$CODE + 0962

```
1149 1273 1 GLOBAL ROUTINE DBG$PREVLOC(PRM_DESC) =
1150 1274 1
1151 1275 1 FUNCTION
1152 1276 1     This routine finds the "previous location", denoted in the command
1153 1277 1     language as %PREVLOC or %. It accepts a Primary Descriptor for the
1154 1278 1     current location as input and returns either a Primary Descriptor or
1155 1279 1     a Volatile Value Descriptor for the previous location as output. If
1156 1280 1     the current location is a structured object of some sort (like an
1157 1281 1     array), MODIFY PRIMARY is called to find the logical predecessor and
1158 1282 1     the modified Primary Descriptor is returned. Otherwise, this routine
1159 1283 1     determines the previous instruction location or the previous data
1160 1284 1     location and returns a Volatile Value Descriptor for that location.
1161 1285 1
1162 1286 1 INPUTS
1163 1287 1     PRM_DESC - A pointer to the input Primary Descriptor for the location
1164 1288 1     whose logical predecessor is to be computed.
1165 1289 1
1166 1290 1 OUTPUTS
1167 1291 1     A pointer to the Primary Descriptor or Volatile Value Descriptor for
1168 1292 1     the logical predecessor location is returned as this
1169 1293 1     routine's value.
1170 1294 1
1171 1295 1 BEGIN
1172 1296 1
1173 1297 1 MAP
1174 1298 1     PRM_DESC: REF DBG$PRIMARY;           ! Pointer to Primary Descriptor
1175 1299 1
1176 1300 1 LOCAL
1177 1301 1     ADDRESS,           ! Address of the current location - 1
1178 1302 1     DUMMY,             ! Dummy routine argument
1179 1303 1     LENGTH,
1180 1304 1     LINE,              ! Line number of the last instruction
1181 1305 1     NEW_ADDR,          ! Address of the current instruction
1182 1306 1     OLD_ADDR,          ! Address of previous instruction
1183 1307 1     PC_BEG,            ! Beginning PC of current source line
1184 1308 1     PC_END,            ! Ending PC of current source line
1185 1309 1     STATUS,
1186 1310 1     STMT,              ! Statement number of last instruction
1187 1311 1     SYMID: REF RST$ENTRY, ! The SYMID of the nearest preceding
1188 1312 1     ! symbol (used for instructions)
1189 1313 1
1190 1314 1     REG_DESC: DBG$REGDESCR,
1191 1315 1     VAL_DESC: REF DBG$VALDESC;         ! Pointer to returned Value Descriptor
1192 1316 1
1193 1317 1
1194 1318 1
1195 1319 1 ! If the input Primary Descriptor describes a structure object, like an
1196 1320 1 ! array or record, let MODIFY PRIMARY modify the Primary Descriptor to
1197 1321 1 ! describe the logical predecessor. Then return that Primary.
1198 1322 1
1199 1323 1 STATUS = MODIFY_PRIMARY(.PRM_DESC, 1);
1200 1324 1 IF .STATUS THEN RETURN .PRM_DESC;
1201 1325 1
1202 1326 1
1203 1327 1 ! If there is a defined "current location" (%CURLOC), then use the Primary
1204 1328 1 ! or Value Descriptor for that entity to set up a Volatile Value Descriptor
1205 1329 1 ! for the previous location.
```

```
1206 1330
1207 1331
1208 1332
1209 1333
1210 1334
1211 1335
1212 1336
1213 1337
1214 1338
1215 1339
1216 1340
1217 1341
1218 1342
1219 1343
1220 1344
1221 1345
1222 1346
1223 1347
1224 1348
1225 1349
1226 1350
1227 1351
1228 1352
1229 1353
1230 1354
1231 1355
1232 1356
1233 1357
1234 1358
1235 1359
1236 1360
1237 1361
1238 1362
1239 1363
1240 1364
1241 1365
1242 1366
1243 1367
1244 1368
1245 1369
1246 1370
1247 1371
1248 1372
1249 1373
1250 1374
1251 1375
1252 1376
1253 1377
1254 1378
1255 1379
1256 1380
1257 1381
1258 1382
1259 1383
1260 1384
1261 1385
1262 1386

! IF .DBG$GL_CURLOC_VMSDESC NEQ 0
THEN
  BEGIN
    VAL_DESC = DBG$MAKE_VAL_DESC(.DBG$GL_CURLOC_VMSDESC, DBG$K_V_VALUE_DESC);
    VAL_DESC[DBG$B_DHDR_LANG] = .PRM_DESC[DBG$B_DHDR_LANG];
    VAL_DESC[DBG$B_DHDR_SYMID0] = .PRM_DESC[DBG$B_DHDR_SYMID0];
  END

! But if no current location is defined, give an error message or use the
! input Primary Descriptor to set up the previous location descriptor.
ELSE
  BEGIN
    IF .STATUS EQL 2 THEN SIGNAL(DBG$NOPRED);
    DBG$PRIM_TO_VAL(.PRM_DESC, DBG$K_V_VALUE_DESC, VAL_DESC);
  END;

! There is no logical successor for an unaligned bit string, so for that
! case we signal an error.
IF .VAL_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
THEN
  SIGNAL(DBG$NOPRED);

! -----
LENGTH = (DBG$DATA_LENGTH(VAL_DESC[DBG$A_VALUE_VMSDESC]) - 1)/XBPUNIT + 1;
REG_DESC = DBG$STA_ADDRESS_TO_REGDESCR(.VAL_DESC[DBG$B_VALUE_POINTER]);
IF (.REG_DESC NEQ 0) AND (.REG_DESC<W> LSS0 (X'00B4' + .DBG$GW_DFLTLENG))
THEN
  SIGNAL(DBG$NOPRED);

! Initialize the DTYPE of the logical predecessor to be type Z (unknown)
! and assume its address is one byte before the current location. This
! may get changed below if appropriate.
VAL_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_Z;
ADDRESS = .VAL_DESC[DBG$B_VALUE_POINTER] - 1;

! If the type of the current object is instruction or entry mask, try to
! locate the previous instruction.
IF (.VAL_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZI) OR
(.VAL_DESC[DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_ZEM)
THEN
  BEGIN
    OLD_ADDR = .ADDRESS;

! First try to symbolize the current location - 1 byte (the contents of
! ADDRESS) to find the nearest routine, block, or label preceding the
```

```

1263 1387 3
1264 1388 3
1265 1389 3
1266 1390 3
1267 1391 3
1268 1392 4
1269 1393 3
1270 1394 4
1271 1395 4
1272 1396 4
1273 1397 4
1274 1398 4
1275 1399 4
1276 1400 4
1277 1401 4
1278 1402 4
1279 1403 4
1280 1404 4
1281 1405 3
1282 1406 4
1283 1407 3
1284 1408 3
1285 1409 3
1286 1410 3
1287 1411 3
1288 1412 3
1289 1413 3
1290 1414 3
1291 1415 4
1292 1416 4
1293 1417 3
1294 1418 3
1295 1419 3
1296 1420 3
1297 1421 3
1298 1422 3
1299 1423 3
1300 1424 3
1301 1425 4
1302 1426 4
1303 1427 4
1304 1428 4
1305 1429 3
1306 1430 3
1307 1431 3
1308 1432 3
1309 1433 3
1310 1434 3
1311 1435 3
1312 1436 3
1313 1437 3
1314 1438 3
1315 1439 3
1316 1440 3
1317 1441 3
1318 1442 3
1319 1443 3

```

```

! current instruction. If no such symbol is found, we just leave the
! address as the current instruction address - 1; there is no way we can
! locate the true previous instruction.

```

```

STATUS = DBG$PC_TO_SYMID(.ADDRESS, SYMID);
IF .STATUS AND T.SYMID NEQ 0)
THEN

```

```

    BEGIN

```

```

    ! A symbol preceding the current instruction was found. If this
    ! is an instruction symbol (a routine, block, or label), save its
    ! address for the forward scan to the desired instruction. However,
    ! if a line number preceding the current instruction can be found,
    ! use that address instead for a shorter forward scan.

```

```

    IF (.SYMID[RST$B_KIND] EQL RST$K_ROUTINE) OR
       (.SYMID[RST$B_KIND] EQL RST$K_BLOCK) OR
       (.SYMID[RST$B_KIND] EQL RST$K_LABEL)
    THEN

```

```

        BEGIN

```

```

            OLD_ADDR = .SYMID[RST$L_STARTADDR];

```

```

            DUMMY = .SYMID;

```

```

            IF DBG$PC_TO_LINE_LOOKUP(.ADDRESS,
                                     LINE, STMT, PC_BEG, PC_END, DUMMY)
            THEN

```

```

                OLD_ADDR = .PC_BEG;

```

```

            END;

```

```

        END;

```

```

        ! End of code if we found a symbolization

```

```

! We now some address where to start the forward scan that looks for
! the previous instruction. Scan forward from that address until the
! desired previous instruction is found.

```

```

WHILE TRUE DO

```

```

    BEGIN

```

```

        NEW_ADDR = DBG$INS_DECODE(.OLD_ADDR, FALSE);

```

```

        IF .NEW_ADDR GTRA .ADDRESS THEN EXITLOOP;

```

```

        OLD_ADDR = .NEW_ADDR;

```

```

    END;

```

```

! Fill the address and length of the found previous instruction into
! the Value Descriptor. Also determine if this location is an entry
! mask--if so, set the DTYPE to be ZEM instead of ZI.

```

```

VAL_DESC[DBG$L_VALUE_POINTER] = .OLD_ADDR;

```

```

VAL_DESC[DBG$W_VALUE_LENGTH] = .NEW_ADDR - .OLD_ADDR;

```

```

VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;

```

```

IF DBG$IS_IT_ENTRY(.OLD_ADDR)
THEN

```

```

    VAL_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZEM;

```

```

END

```

```

! End of code for previous instruction

```



1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341

1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465

! The current location is not an instruction location. We therefore set  
! up the Value Descriptor for a data object.

ELSE  
BEGIN  
VAL\_DESC[DBG\$B\_DHDR\_FCODE] = RST\$K\_TYPE\_DESCR;  
VAL\_DESC[DBG\$V\_DHDR\_FORMAT] = 0;  
VAL\_DESC[DBG\$B\_VALUE\_CLASS] = DSC\$K\_CLASS\_S;  
VAL\_DESC[DBG\$B\_VALUE\_DTYPE] = .DBG\$G\_DFLTTP;  
VAL\_DESC[DBG\$W\_VALUE\_LENGTH] = .DBG\$G\_DFLTLENG;  
VAL\_DESC[DBG\$L\_VALUE\_POINTER] = .VAL\_DESC[DBG\$L\_VALUE\_POINTER] - .LENGTH;  
END;

! Return a pointer to the Volatile Value Descriptor for the previous  
! location.

RETURN .VAL\_DESC;

END;

59	00000000G	00	9E	00002	.ENTRY	DBG\$PREVLOC, Save R2,R3,R4,R5,R6,R7,R8,R9	1273
58	00000000G	00	9E	00009	MOVAB	DBG\$GW_DFLTLENG, R9	
5E		1C	C2	00010	MOVAB	LIB\$SIGNAL, R8	
		01	DD	00013	SUBL2	#28, SP	
52	04	AC	DD	00015	PUSHL	#1	1323
		52	DD	00019	MOVL	PRM_DESC, R2	
0000V	CF	02	FB	0001B	PUSHL	R2	
57		50	DD	00020	CALLS	#2, MODIFY_PRIMARY	
03		57	E9	00023	MOVL	R0, STATUS	
		014D	31	00026	BLBC	STATUS, 1\$	1324
50	00000000G	00	DD	00029	BRW	12\$	
		1C	13	00030	MOVL	DBG\$GL_CURLOC_VMSDESC, R0	1331
7E	83	8F	9A	00032	BEQL	2\$	
		50	DD	00036	MOVZBL	#131, -(SP)	1334
00000000G	00	02	FB	00038	PUSHL	R0	
		50	DD	0003F	CALLS	#2, DBG\$MAKE_VAL_DESC	
03	A0	03	A2	90	MOVL	R0, VAL_DESC	1335
0C	A0	0C	A2	DD	MOVB	3(R2), 3(R0)	1336
		1D	11	0004C	MOVL	12(R2), 12(R0)	1331
02		57	D1	0004E	BRB	4\$	1345
		09	12	00051	CMPL	STATUS, #2	
	00028810	8F	DD	00053	BNEQ	3\$	
68		01	FB	00059	PUSHL	#165904	
		5E	DD	0005C	CALLS	#1, LIB\$SIGNAL	
7E	83	8F	9A	0005E	PUSHL	SP	1346
		52	DD	00062	MOVZBL	#131, -(SP)	
00000000G	00	03	FB	00064	PUSHL	R2	
		6E	DD	0006B	CALLS	#3, DBG\$PRIM_TO_VAL	
54	14	A2	9E	0006E	MOVL	VAL_DESC, R2	1353
					MOVAB	20(R2), R4	

		0D	03	A4	91	00072	CMPB	3(R4), #13		
				09	12	00076	BNEQ	5\$		
			00028810	8F	DD	00078	PUSHL	#165904	1355	
		68		01	FB	0007E	CALLS	#1, LIB\$SIGNAL		
				54	DD	00081	PUSHL	R4	1360	
		00000000G	00	01	FB	00083	CALLS	#1, DBG\$DATA_LENGTH		
				50	D7	0008A	DECL	R0		
		50		08	C6	0008C	DIVL2	#8, R0		
		56	01	A0	9E	0008F	MOVAB	1(R0), LENGTH		
			18	A2	DD	00093	PUSHL	24(R2)	1361	
		00000000G	00	01	FB	00096	CALLS	#1, DBG\$STA_ADDRESS_TO_REGDESCR		
				50	D5	0009D	TSTL	REG_DESC	1362	
				18	13	0009F	BEQL	6\$		
		51		69	3C	000A1	MOVZWL	DBG\$GW_DFLTLENG, R1		
		51	00B4	C1	9E	000A4	MOVAB	180(R1), R1		
51	50	10		00	ED	000A9	CMPZV	#0, #16, REG_DESC, R1		
				09	1E	000AE	BGEQU	6\$		
			00028810	8F	DD	000B0	PUSHL	#165904	1364	
		68		01	FB	000B6	CALLS	#1, LIB\$SIGNAL		
			03	A4	94	000B9	CLRB	3(R4)	1371	
		55	18	01	C3	000BC	SUBL3	#1, 24(R2), ADDRESS	1372	
				02	A4	91	000C1	CMPB	2(R4), #22	1378
				09	13	000C5	BEQL	7\$		
		17	02	A4	91	000C7	CMPB	2(R4), #23	1379	
				03	13	000CB	BEQL	7\$		
			008A	31	000CD	BRW	11\$			
		53		55	DD	000DD	MOVL	ADDRESS, OLD_ADDR	1382	
			04	AE	9F	000D3	PUSHAB	SYMID	1391	
				55	DD	000D6	PUSHL	ADDRESS		
		00000000G	00	02	FB	000D8	CALLS	#2, DBG\$PC_TO_SYMID		
			57	50	DD	000DF	MOVL	R0, STATUS		
		42		57	E9	000E2	BLBC	STATUS, 9\$	1392	
			04	AE	D5	000E5	TSTL	SYMID		
				3D	13	000E8	BEQL	9\$		
		50	04	AE	DD	000EA	MOVL	SYMID, R0	1403	
		02	14	A0	91	000EE	CMPB	20(R0), #2		
				0C	13	000F2	BEQL	8\$		
		03	14	A0	91	000F4	CMPB	20(R0), #3	1404	
				06	13	000F8	BEQL	8\$		
		04	14	A0	91	000FA	CMPB	20(R0), #4	1405	
				27	12	000FE	BNEQ	9\$		
		53	18	A0	DD	00100	MOVL	24(R0), OLD_ADDR	1408	
08	AE			50	DD	00104	MOVL	R0, DUMMY	1409	
			08	AE	9F	00108	PUSHAB	DUMMY	1410	
			10	AE	9F	0010B	PUSHAB	PC_END		
			18	AE	9F	0010E	PUSHAB	PC_BEG		
			20	AE	9F	00111	PUSHAB	STAT		
			28	AE	9F	00114	PUSHAB	LINE		
				55	DD	00117	PUSHL	ADDRESS		
		00000000G	00	06	FB	00119	CALLS	#6, DBG\$PC_TO_LINE_LOOKUP		
			04	50	E9	00120	BLBC	R0, 9\$		
		53	10	AE	DD	00123	MOVL	PC_BEG, OLD_ADDR	1413	
				7E	D4	00127	CLRL	-(SP)	1426	
				53	DD	00129	PUSHL	OLD_ADDR		
		00000000G	00	02	FB	0012B	CALLS	#2, DBG\$INS_DECODE		
			55	50	D1	00132	CMPB	NEW_ADDR, ADDRESS	1427	
				05	1A	00135	BGTRU	10\$		

		53		50	D0 00137		MOVL	NEW_ADDR, OLD_ADDR	:	1428
				EB	11 0013A		BRB	9\$	:	1424
64	18	A2		53	D0 0013C	10\$:	MOVL	OLD_ADDR, 24(R2)	:	1436
		50		53	A3 00140		SUBW3	OLD_ADDR, NEW_ADDR, (R4)	:	1437
	02	A4		16	90 00144		MOVB	#22, 2(R4)	:	1438
				53	D0 00148		PUSHL	OLD_ADDR	:	1439
00000000G	00			01	FB 0014A		CALLS	#1, DBG\$IS_IT_ENTRY	:	
	22			50	E9 00151		BLBC	R0, 12\$	:	
	02	A4		17	90 00154		MOVB	#23, 2(R4)	:	1441
				1C	11 00158		BRB	12\$	:	1378
	06	A2		03	90 0015A	11\$:	MOVB	#3, 6(R2)	:	1451
	05	A2	F0	8F	8A 0015E		BICB2	#240, 5(R2)	:	1452
	03	A4		01	90 00163		MOVB	#1, 3(R4)	:	1453
	02	A4	00000000G	00	90 00167		MOVB	DBG\$GL_DFLTTP, 2(R4)	:	1454
		64		69	B0 0016F		MOVW	DBG\$GW_DFLTLENG, (R4)	:	1455
	18	A2		56	C2 00172		SUBL2	LENGTH, 24(R2)	:	1456
		50		52	D0 00176	12\$:	MOVL	R2, R0	:	1463
					04 00179		RET		:	1465

; Routine Size: 378 bytes, Routine Base: DBG\$CODE + 0A96

```

1343 1466 1 ROUTINE MODIFY_PRIMARY(PRM_DESC: REF DBG$PRIMARY, DIRECTION) =
1344 1467 1
1345 1468 1 FUNCTION
1346 1469 1     This routine takes a Primary Descriptor and attempts to find
1347 1470 1     the logical successor (DIRECTION=0) or logical predecessor
1348 1471 1     (DIRECTION=1). For example, the Primary for X(0) might
1349 1472 1     be modified to be a Primary for X(1).
1350 1473 1
1351 1474 1 INPUTS
1352 1475 1     PRM_DESC - points to a Primary Descriptor
1353 1476 1     DIRECTION - if 0, we want the logical successor.
1354 1477 1                 if 1, we want the logical predecessor.
1355 1478 1
1356 1479 1 OUTPUTS
1357 1480 1     1 is returned if this routine is successful, and
1358 1481 1     the Primary pointed to by PRM_DESC is modified.
1359 1482 1     If this routine fails, it returns 0 or 2.
1360 1483 1
1361 1484 1
1362 1485 1 BEGIN
1363 1486 1
1364 1487 1 MAP
1365 1488 1     PRM_DESC: REF DBG$PRIMARY;      ! Pointer to input Primary Descriptor
1366 1489 1
1367 1490 1 BUILTIN
1368 1491 1     INSQUE,
1369 1492 1     REMQUE;
1370 1493 1
1371 1494 1 LABEL
1372 1495 1     PASS,
1373 1496 1     SCAN;
1374 1497 1
1375 1498 1 LOCAL
1376 1499 1     COMP_FLAG: BYTE,
1377 1500 1     ERROR_STATUS,
1378 1501 1     DUMMY,
1379 1502 1     ROOT_ADR,
1380 1503 1     SUB_NODE: REF DBG$PRIM_NODE,
1381 1504 1     SYM_NAME: REF VECTOR(.BYTE);
1382 1505 1
1383 1506 1
1384 1507 1
1385 1508 1     ! Save away the "current" primary. This is used by our stack
1386 1509 1     ! machine code in RSTACCESS to evaluate the "push inner record
1387 1510 1     ! address" and "push outer record address" instructions.
1388 1511 1
1389 1512 1     DBG$GL_CURRENT_PRIMARY = .PRM_DESC;
1390 1513 1
1391 1514 1     ! Give up if the descriptor is not a Primary we can modify
1392 1515 1     ! to get logical successor or predecessor.
1393 1516 1
1394 1517 1     IF (.PRM_DESC[DBG$B_DHDR_TYPE] NEQ DBG$K_PRIMARY_DESC) OR
1395 1518 1         ((.PRM_DESC[DBG$B_DHDR_KIND] NEQ RST$K_DATA) AND
1396 1519 1         (.PRM_DESC[DBG$B_DHDR_KIND] NEQ RST$K_TYPCOMP)) OR
1397 1520 1         (.PRM_DESC[DBG$V_DHDR_SUBREF])
1398 1521 1     THEN
1399 1522 1         RETURN 0;

```





```

1457 1580 7
1458 1581 7
1459 1582 8
1460 1583 8
1461 1584 8
1462 1585 8
1463 1586 8
1464 1587 8
1465 1588 8
1466 1589 8
1467 1590 8
1468 1591 8
1469 1592 8
1470 1593 8
1471 1594 8
1472 1595 8
1473 1596 8
1474 1597 8
1475 1598 8
1476 1599 8
1477 1600 8
1478 1601 8
1479 1602 8
1480 1603 8
1481 1604 8
1482 1605 8
1483 1606 8
1484 1607 8
1485 1608 9
1486 1609 8
1487 1610 9
1488 1611 9
1489 1612 9
1490 1613 9
1491 1614 9
1492 1615 8
1493 1616 8
1494 1617 8
1495 1618 8
1496 1619 8
1497 1620 8
1498 1621 7
1499 1622 8
1500 1623 8
1501 1624 8
1502 1625 8
1503 1626 8
1504 1627 8
1505 1628 7
1506 1629 7
1507 1630 6
1508 1631 6
1509 1632 6
1510 1633 6
1511 1634 7
1512 1635 7
1513 1636 7

```

```

IF .S_VECT[.S,DBG$PNSUB_SVALUE] EQL .S_VECT[.S,DBG$PNSUB_UBOUND]
THEN
  BEGIN
    ! If we have no more dimensions and we
    ! are at the top subnode (i.e., there are
    ! no "higher" levels at which we can
    ! increment something) then go ahead
    ! and increment it, (but giving a warning
    ! that we are at the upper bound).
    ! For example, if X is a one-dimensional
    ! array from 1 to 3, and we want the logical
    ! successor of X(3), we'll go ahead and return
    ! X(4) but we'll give an informational saying
    ! you have walked past the upper bound.
    ! But if X were 2-dimensional, say 1:3 by 1:3,
    ! and you want the successor of X(1,3),
    ! then return X(2,1) and not X(1,4).
    ! Or if X were a record of arrays,
    ! and X.A(3) was the upper bound, then
    ! you would want to go to the next
    ! record component, say X.B, instead of
    ! going to X.A(4).
    ! That is the reason for the checks for
    ! DIMENSION EQL DIMCNT and BLINK EQL
    ! ROOT_ADR.
    IF (.DIMENSION EQL .SUB_NODE[DBG$B PNARR DIMCNT]) AND
    (.SUB_NODE[DBG$PNSUB_BLINK] EQL .ROOT_ADR)
    THEN
      BEGIN
        SIGNAL(DBG$SUBSCRNG, 3, UPLIT BYTE(ASCII 'upper'), .DIMENSION, .S_VECT
        S_VECT[.S,DBG$PNSUB_SVALUE] = .S_VECT[.S,DBG$PNSUB_SVALUE] + T;
        LEAVE SCAN;
      END
    ELSE
      ! Set back to lower bound.
      S_VECT[.S,DBG$PNSUB_SVALUE] = .S_VECT[.S,DBG$PNSUB_LBOUND]
    END
  ELSE
    BEGIN
      ! Increment and leave loop.
      S_VECT[.S,DBG$PNSUB_SVALUE] = .S_VECT[.S,DBG$PNSUB_SVALUE] + 1;
      LEAVE SCAN;
    END
  END
ELSE
  ! Logical predecessor.
  BEGIN
    IF .S_VECT[.S,DBG$PNSUB_SVALUE] EQL .S_VECT[.S,DBG$PNSUB_LBOUND]
    THEN

```

```

1514 1637
1515 1638
1516 1639
1517 1640
1518 1641
1519 1642
1520 1643
1521 1644
1522 1645
1523 1646
1524 1647
1525 1648
1526 1649
1527 1650
1528 1651
1529 1652
1530 1653
1531 1654
1532 1655
1533 1656
1534 1657
1535 1658
1536 1659
1537 1660
1538 1661
1539 1662
1540 1663
1541 1664
1542 1665
1543 1666
1544 1667
1545 1668
1546 1669
1547 1670
1548 1671
1549 1672
1550 1673
1551 1674
1552 1675
1553 1676
1554 1677
1555 1678
1556 1679
1557 1680
1558 1681
1559 1682
1560 1683
1561 1684
1562 1685
1563 1686
1564 1687
1565 1688
1566 1689
1567 1690
1568 1691
1569 1692
1570 1693

```

```

BEGIN
    ! If we have no more dimensions then go ahead
    ! and decrement it, (but giving a warning
    ! that we are at the lower bound).
    ! For example, if X is a one-dimensional
    ! array from 1 to 3, and we want the logical
    ! predecessor of X(1), we'll go ahead and return
    ! X(0) but we'll give an informational saying
    ! you have walked past the upper bound.
    ! But if X were 2-dimensional, say 1:3 by 1:3,
    ! and you want the predecessor of X(3,1),
    ! then return X(2,3) and not X(3,0).
    ! That is the reason for this check for
    ! DIMENSION EQL DIMCNT.
    IF (.DIMENSION EQL .SUB_NODE[DBG$B_PNARR_DIMCNT]) AND
       (.SUB_NODE[DBG$L_PNODE_BLINK] EQL .ROOT_ADR)
    THEN
        BEGIN
            SIGNAL(DBG$ SUBSCRNG, 3, UPLIT BYTE(ASCII('lower'), .DIMENSION, .S_VECT
            S_VECT[.S,DBG$L_PNSUB_SVALUE] = .S_VECT[.S,DBG$L_PNSUB_SVALUE] - 1;
            LEAVE SCAN;
        END
    ELSE
        ! Set back to upper bound.
        !
        ! S_VECT[.S,DBG$L_PNSUB_SVALUE] = .S_VECT[.S,DBG$L_PNSUB_UBOUND]
    END
    ELSE
        BEGIN
            ! Decrement and leave loop.
            !
            ! S_VECT[.S,DBG$L_PNSUB_SVALUE] = .S_VECT[.S,DBG$L_PNSUB_SVALUE] - 1;
            LEAVE SCAN;
        END;
    END;
END;

[RST$K_TYPE_RECORD,RST$K_TYPE_VARIANT]:
BEGIN
    ERROR STATUS = 2;
    IF .DIRECTION EQL 0          ! 0 = NEXTLOC, 1 = PREVLOC
    THEN
        ! Logical successor.
        !
        BEGIN
            ! If we can go to the next component, do so and exit
            ! the loop.
            !
            IF .SUB_NODE[DBG$W_PNREC_INDEX] LSSU .SUB_NODE[DBG$W_PNREC_NCOMPS]
            THEN

```

```

1571 1694 7 BEGIN
1572 1695 7 SUB_NODE[DBG$W_PNREC_INDEX] = .SUB_NODE[DBG$W_PNREC_INDEX] + 1;
1573 1696 7 LEAVE SCAN;
1574 1697 6 END;
1575 1698 6 END
1576 1699 6 ELSE
1577 1700 6
1578 1701 5 ! Logical predecessor.
1579 1702 5
1580 1703 6 BEGIN
1581 1704 6
1582 1705 6 ! If we can get to the previous component, do so
1583 1706 6 and exit the loop.
1584 1707 6
1585 1708 6 IF .SUB_NODE[DBG$W_PNREC_INDEX] GTRU 1
1586 1709 6 THEN
1587 1710 7 BEGIN
1588 1711 7 SUB_NODE[DBG$W_PNREC_INDEX] = .SUB_NODE[DBG$W_PNREC_INDEX] - 1;
1589 1712 7 LEAVE SCAN;
1590 1713 7 END
1591 1714 5 END;
1592 1715 5
1593 1716 4 [OTHERWISE]:0;
1594 1717 4 TES;
1595 1718 4
1596 1719 4 ! If we fall through to here without succeeding in incrementing
1597 1720 4 or decrementing anything, then error status will still be
1598 1721 4 0 or 2 and we return it, indicating we did not succeed.
1599 1722 4
1600 1723 4 RETURN .ERROR_STATUS;
1601 1724 3 END; ! End of block scan
1602 1725 3
1603 1726 3
1604 1727 3
1605 1728 3
1606 1729 3
1607 1730 3
1608 1731 3
1609 1732 4
1610 1733 3 IF (.SUB_NODE[DBG$L_PNODE_FLINK] EQLA .PRM_DESC[DBG$L_PRIM_BLINK])
1611 1734 4 AND
1612 1735 4 (.SUB_NODE[DBG$B_PNODE_FCODE] EQL RST$K_TYPE_ARRAY)
1613 1736 4
1614 1737 4 THEN RETURN 1;
1615 1738 4
1616 1739 4
1617 1740 4 ! We have found the composite entry we are going to modify. Strip off
1618 1741 4 all subsequent primary sub-nodes, and then add new sub_nodes to get
1619 1742 4 a primary which describes a single data item.
1620 1743 4
1621 1744 4 WHILE .SUB_NODE[DBG$L_PNODE_FLINK] NEQA .ROOT_ADR DO
1622 1745 4 REMQUET.SUB_NODE[DBG$L_PNODE_FLINK],DUMMY;
1623 1746 3
1624 1747 4 IF .PRM_DESC[DBG$V_DHDR_TMPREF] THEN
1625 1748 4 BEGIN
1626 1749 4 PRM_DESC[DBG$V_DHDR_TMPREF] = FALSE;
1627 1750 4 PRM_DESC[DBG$V_DHDR_SUBREF] = FALSE;
1627 1750 4 PRM_DESC[DBG$W_PRIM_OFFSET] = 0;

```



```

1628 1751 4      PRM_DESC(DBG$W_PRIM_LENGTH) = 0;
1629 1752 4      END;
1630 1753 4
1631 1754 4      COMP_FLAG = FALSE;
1632 1755 4
1633 1756 4      WHILE TRUE DO
1634 1757 4      BEGIN
1635 1758 4      LOCAL SYMID,TYPEID,FCODE,KIND;
1636 1759 4      SELECTONE .SUB_NODE(DBG$B_PNODE_FCODE) OF
1637 1760 4      SET
1638 1761 4      [RST$K_TYPE_ARRAY]:
1639 1762 5      BEGIN
1640 1763 5      IF .COMP_FLAG THEN
1641 1764 6      BEGIN
1642 1765 6      LOCAL S_VECT : REF DBG$PRIM_NODE SUBS;
1643 1766 6      S_VECT = SUB_NODE(DBG$A_PNARR_SVECTOR);
1644 1767 6      DECR S FROM .SUB_NODE(DBG$B_PNARR_DIMCNT)-1 TO 0 DO
1645 1768 6      S_VECT[S,DBG$C_PNSUB_SVALUE] = .S_VECT[S,DBG$L_PNSUB_UBOUND];
1646 1769 5      END;
1647 1770 5      SYMID = 0;
1648 1771 5      KIND = RST$K_DATA;
1649 1772 5      DBG$STA_SYMTYPE(.SUB_NODE(DBG$L_PNARR_CELLTYPE),FCODE,TYPEID);
1650 1773 4      END;
1651 1774 4
1652 1775 4      [RST$K_TYPE_RECORD,RST$K_TYPE_VARIANT]:
1653 1776 5      BEGIN
1654 1777 5      LOCAL N_COMPS,S_VECT : REF VECTOR[,LONG];
1655 1778 5      IF .SUB_NODE(DBG$B_PNODE_FCODE) EQL RST$K_TYPE_RECORD
1656 1779 5      THEN DBG$STA_TYP_RECORD(.SUB_NODE(DBG$L_PNODE_TYPEID),N_COMPS,S_VECT,DUMMY)
1657 1780 5      ELSE
1658 1781 6      BEGIN
1659 1782 6      +
1660 1783 6      We have a variant set. If the TYPEID field is non-zero this is
1661 1784 6      a new variant sub node (added by us on the last pass), and so we
1662 1785 6      need to obtain the current value of the TAG field. A zero-length
1663 1786 6      tag field (or an illegal tag value) cause this entire variant to
1664 1787 6      be ignored.
1665 1788 6
1666 1789 6      IF (TYPEID = .SUB_NODE(DBG$L_PNODE_TYPEID)) NEQ 0 THEN
1667 1790 7      BEGIN
1668 1791 7      LOCAL TAG,MARK,TYPE,CODE,
1669 1792 7      VAL_DESC : REF DBG$VALDESC,
1670 1793 7      VARIANT : REF RST$VAR_ENTRY;
1671 1794 7      MAP TYPEID : REF RST$ENTRY;
1672 1795 7
1673 1796 7      REMQUE(.SUB_NODE,SUB_NODE);
1674 1797 7      IF (TAG = .TYPEID[RST$L_VARTAGPTR]) EQL 0 THEN LEAVE PASS;
1675 1798 7      DBG$STA_SYMTYPE(.TAG,SYM_NAME);
1676 1799 7      IF .SYM_NAME[0] EQL 0 THEN LEAVE PASS;
1677 1800 7      +
1678 1801 7      We now need to obtain the actual tag value. This is done
1679 1802 7      by stripping off the VARIANT sub-node, and adding a new
1680 1803 7      primary sub-node describing the Tag field. We then call
1681 1804 7      dbg$prim_to_val to get a Debug Value descriptor, extract
1682 1805 7      the tag value, and restore the state of the primary.
1683 1806 7      -
1684 1807 7      MARK = DBG$PUSH_TEMPMEH();

```

```

1685 1808 7
1686 1809 7
1687 1810 7
1688 1811 7
1689 1812 7
1690 1813 7
1691 1814 7
1692 1815 7
1693 1816 7
1694 1817 7
1695 1818 7
1696 1819 7
1697 1820 7
1698 1821 7
1699 1822 7
1700 1823 6
1701 1824 6
1702 1825 6
1703 1826 5
1704 1827 5
1705 1828 5
1706 1829 5
1707 1830 5
1708 1831 5
1709 1832 6
1710 1833 6
1711 1834 6
1712 1835 6
1713 1836 6
1714 1837 5
1715 1838 5
1716 1839 4
1717 1840 4
1718 1841 4
1719 1842 4
1720 1843 4
1721 1844 4
1722 1845 4
1723 1846 4
1724 1847 4
1725 1848 4
1726 1849 4
1727 1850 4
1728 1851 4
1729 1852 4
1730 1853 4
1731 1854 4
1732 1855 4
1733 1856 1

```

```

DBG$STA SYMTYPE(.TAG, CODE, TYPE);
DBG$BUILD PRIMARY SUBNODE(.PRM DESC, RST$K DATA, .TAG, .CODE, .TYPE, 0);
DBG$PRIM TO VAL(.PRM DESC, DBG$K VALUE_DESC, VAL_DESC);
TAG = .VAL DESC(DBG$K VALUE_VALDEQ);
REMOVE(.PRM DESC(DBG$K PRIM_BLINK), DUMMY);
DBG$POP TEMPHEM(.MARK);
IF (VARIANT = DBG$STA VARIANT_SELECT(.TAG, .TYPEID)) EQL 0 THEN LEAVE PASS;
SUB_NODE(DBG$V_PNVAR_VALID) = TRUE;
SUB_NODE(DBG$W_PNVAR_INDEX) = 1;
SUB_NODE(DBG$L_PNODE_TYPEID) = 0;
SUB_NODE(DBG$L_PNVAR_TAGID) = .TYPEID[RST$L_VARTAGPTR];
SUB_NODE(DBG$W_PNVAR_NCOMPS) = .VARIANT[RST$L_VAR_COMPCNT];
SUB_NODE(DBG$L_PNVAR_COMPLST) = .VARIANT[RST$L_VAR_COMPLST];
SUB_NODE(DBG$L_PNVAR_DSTPTR) = .VARIANT[RST$L_VAR_DSTPTR];
INSQUE(.SUB_NODE, .PRM_DESC(DBG$L_PNVAR_BLINK));
END;
N_COMPS = .SUB_NODE(DBG$W_PNVAR_NCOMPS);
S_VECT = .SUB_NODE(DBG$L_PNVAR_COMPLST);
END;
IF .COMP_FLAG THEN SUB_NODE(DBG$W_PNREC_INDEX) = .N_COMPS;
SYMID = .S_VECT[.SUB_NODE(DBG$W_PNREC_INDEX)-1];
DBG$STA SYMKIND(.SYMID, KIND);
IF .KIND EQL RST$K_VARIANT
THEN
BEGIN
FCODE = RST$K_TYPE_VARIANT;
TYPEID = .SYMID;
SYMID = 0;
END
ELSE
DBG$STA_SYMTYPE(.SYMID, FCODE, TYPEID);
END;
[OTHERWISE]:
EXITLOOP;
TES;
SUB_NODE(DBG$V_PNODE_EVAL) = TRUE;
DBG$BUILD PRIMARY SUBNODE(.PRM DESC, .KIND, .SYMID, .FCODE, .TYPEID, 0);
SUB_NODE = .PRM_DESC(DBG$L_PNVAR_BLINK);
COMP_FLAG = .DIRECTION;
END;
IF .SUB_NODE(DBG$L_PNODE_SYMID) EQL 0 THEN EXITLOOP;
DBG$STA_SYMNAME(.SUB_NODE(DBG$L_PNODE_SYMID), SYM_NAME);
IF .SYM_NAME(0) NEQ 0 THEN EXITLOOP;
END;
RETURN 1;
END;
! End of modify_primary

```

.PSECT DBG\$PLIT, NOWRT, SHR, PIC, 0

```

72 65 70 70 75 05 0003E P.AAJ: .ASCII <5>\upper\
72 65 77 6F 6C 05 00044 P.AAK: .ASCII <5>\lower\

```

:

				.PSECT		DBG\$CODE,NOWRT, SHR, PIC,0	
				OFFC 00000		MODIFY_PRIMARY:	
		5B	00000000G	00	9E	00002	1466
		5E		28	C2	00009	
		50	04	AC	D0	0000C	1512
	00000000G	00		50	D0	00010	
	79	8F	02	A0	91	00017	1517
				11	12	0001C	
		06	07	A0	91	0001E	1518
				06	13	00022	
		0A	07	A0	91	00024	1519
				05	12	00028	
03	04	A0		01	E1	0002A 1\$:	1520
				02DC	31	0002F 2\$:	
				59	D4	00032 3\$:	1524
53	04	AC		14	C1	00034 4\$:	1538
		58		53	D0	00039	
		53	04	A3	D0	0003C 5\$:	1545
		58		53	D1	00040	
				03	12	00043	
				00E3	31	00045	
		F0	0A	A3	E9	00048 6\$:	1546
		50	09	A3	9A	0004C	1547
		01		50	91	00050	1549
				03	13	00053	
				00AA	31	00055	
		59		02	D0	00058 7\$:	1563
		52	28	A3	9E	0005B	1564
		57	1B	A3	9A	0005F	1565
				54	D4	00063	1580
				0091	31	00065	
06	0A	A3		01	E1	00068 8\$:	1567
		55	FF	A4	9E	0006D	1568
				04	11	00071	
55		57		54	C3	00073 9\$:	1569
50		55		14	C5	00077 10\$:	1580
56		52		50	C1	0007B	
			08	AC	D5	0007F	1571
				39	12	00082	
			0C	A042	9F	00084	1580
		9E		66	D1	00088	
				2C	12	0008B	
		57		54	D1	0008D	1607
				21	12	00090	
		58	04	A3	D1	00092	1608
				1B	12	00096	
			0C	A042	9F	00098	1611
				9E	DD	0009C	
				54	DD	0009E	
		00000000'		EF	9F	000A0	
				03	DD	000A6	
		000287AB		8F	DD	000A8	
		6B		05	FB	000AE	
						CALLS #5, LIB\$SIGNAL	

		06	11	000B1	BRB	12\$	1612
	08	A042	9F	000B3	11\$: PUSHAB	8(R0)[S_VECT]	1619
		37	11	000B7	BRB	15\$	
		66	D6	000B9	12\$: INCL	(R6)	1626
		72	11	000B11	BRB	23\$	1627
9E	08	A042	9F	000BD	13\$: PUSHAB	8(R0)[S_VECT]	1635
		66	D1	000C1	CMPL	(R6), @TSP)+	
		2F	12	000C4	BNEQ	16\$	
57		54	D1	000C6	CMPL	DIMENSION, R7	1653
		21	12	000C9	BNEQ	14\$	
58	04	A3	D1	000CB	CMPL	4(SUB_NODE), ROOT_ADR	1654
		1B	12	000CF	BNEQ	14\$	
	0C	A042	9F	000D1	PUSHAB	12(R0)[S_VECT]	1657
		9E	DD	000D5	PUSHL	@(SP)+	
		54	DD	000D7	PUSHL	DIMENSION	
	00000000	EF	9F	000D9	PUSHAB	P.AAK	
		03	DD	000DF	PUSHL	#3	
	000287AB	8F	DD	000E1	PUSHL	#165803	
68		05	FB	000E7	CALLS	#5, LIB\$SIGNAL	
		09	11	000EA	BRB	16\$	1658
	0C	A042	9F	000EC	14\$: PUSHAB	12(R0)[S_VECT]	1665
66		9E	DD	000F0	15\$: MOVL	@(SP)+, (R6)	
		04	11	000F3	BRB	17\$	1637
		66	D7	000F5	16\$: DECL	(R6)	1672
		36	11	000F7	BRB	23\$	1673
01		57	F1	000F9	17\$: ACBL	R7, #1, DIMENSION, 8\$	1565
	FF3A	31	000FF	18\$: BRW	5\$		1547
07		50	91	00102	19\$: CMPB	R0, #7	1679
		05	13	00105	BEQL	20\$	
13		50	91	00107	CMPB	R0, #19	
		F3	12	0010A	BNEQ	18\$	
59		02	DD	0010C	20\$: MOVL	#2, ERROR STATUS	1681
50	18	A3	9E	0010F	MOVAB	24(SUB_NODE), R0	1692
	08	AC	D5	00113	TSTL	DIRECTION	1682
		DA	12	00116	BNEQ	21\$	
02	A0	60	B1	00118	CMPW	(R0), 2(R0)	1692
		E1	1E	0011C	BGEQU	18\$	
		60	B6	0011E	INCW	(R0)	1695
		0D	11	00120	BRB	23\$	1696
01		60	B1	00122	21\$: CMPW	(R0), #1	1708
		D8	18	00125	BLEQU	18\$	
		60	B7	00127	DECW	(R0)	1711
		04	11	00129	BRB	23\$	1712
50		59	DD	0012B	22\$: MOVL	ERROR_STATUS, R0	1723
			04	0012E	RET		
50	04	AC	DD	0012F	23\$: MOVL	PRM_DESC, R0	1732
18	A0	63	D1	00133	CMPL	(SUB_NODE), 24(R0)	
		09	12	00137	BNEQ	24\$	
01	09	A3	91	00139	CMPB	9(SUB_NODE), #1	1734
		03	12	0013D	BNEQ	24\$	
	01C8	31	0013F	BRW	44\$		
58		63	D1	00142	24\$: CMPL	(SUB_NODE), ROOT_ADR	1743
		06	13	00145	BEQL	25\$	
6E	00	B3	OF	00147	REMQUE	@(SUB_NODE), DUMMY	1744
		F5	11	0014B	BRB	24\$	
50	04	AC	DD	0014D	25\$: MOVL	PRM_DESC, R0	1746
09	05	A0	E9	00151	BLBC	5(R0), 26\$	



04	A0	0102	BF	AA	00155	BICW2	#258, 4(R0)	1748	
		10	A0	D4	0015B	CLRL	16(R0)	1750	
			5A	94	0015E	CLRB	COMP FLAG	1754	
	50	09	A3	9A	00160	MOVZBL	9(SUB_NODE), R0	1759	
	01		50	91	00164	CMPB	R0, #1	1761	
			30	12	00167	BNEQ	31\$		
	1B		5A	E9	00169	BLBC	COMP FLAG, 30\$	1763	
	52	28	A3	9E	0016C	MOVAB	40(R3), S_VECT	1766	
	50	1B	A3	9A	00170	MOVZBL	27(SUB_NODE), S	1768	
51			0E	11	00174	BRB	29\$		
	50		14	C5	00176	MULL3	#20, S, R1		
		6142	9F	0017A	PUSHAB	(R1)[S_VECT]			
		0C	A142	9F	0017D	PUSHAB	12(R1)[S_VECT]		
	9E		9E	D0	00181	MOVL	@(SP)+, @(SP)+		
	EF		50	F4	00184	SOBGEQ	S, 28\$		
			55	D4	00187	CLRL	SYMID	1770	
18	AE		06	D0	00189	MOVL	#6, KIND	1771	
		1C	AE	9F	0018D	PUSHAB	TYPEID	1772	
		24	AE	9F	00190	PUSHAB	FCODE		
		24	A3	DD	00193	PUSHL	36(SUB_NODE)		
			0127	31	00196	BRW	41\$		
	07		50	91	00199	CMPB	R0, #7	1775	
			08	13	0019C	BEQL	32\$		
	13		50	91	0019E	CMPB	R0, #19		
			03	13	001A1	BEQL	32\$		
		014A	31	001A3	BRW	43\$			
	07		50	91	001A6	CMPB	R0, #7	1778	
			15	12	001A9	BNEQ	33\$		
			5E	DD	001AB	PUSHL	SP	1779	
		08	AE	9F	001AD	PUSHAB	S_VECT		
		10	AE	9F	001B0	PUSHAB	N_COMPS		
		0C	A3	DD	001B3	PUSHL	12(SUB_NODE)		
00000000G	00		04	FB	001B6	CALLS	#4, DBG\$STA_TYP_RECORD		
		00C5	31	001BD	BRW	38\$			
	1C	AE	0C	A3	D0	001C0	MOVL	12(SUB_NODE), TYPEID	1789
			03	12	001C5	BNEQ	34\$		
		00B1	31	001C7	BRW	37\$			
	53		63	0F	001CA	REMQUE	(SUB_NODE), SUB_NODE	1796	
	50	1C	AE	D0	001CD	MOVL	TYPEID, R0	1797	
	54	10	A0	D0	001D1	MOVL	16(R0), TAG		
			79	13	001D5	BEQL	35\$		
		24	AE	9F	001D7	PUSHAB	SYM_NAME	1798	
			54	DD	001DA	PUSHL	TAG		
00000000G	00		02	FB	001DC	CALLS	#2, DBG\$STA_SYMNAME		
		24	BE	95	001E3	TSTB	@SYM_NAME	1799	
			68	13	001E6	BEQL	35\$		
00000000G	00		00	FB	001E8	CALLS	#0, DBG\$PUSH_TEMPMEM	1807	
	57		50	D0	001EF	MOVL	R0, MARK		
		0C	AE	9F	001F2	PUSHAB	TYPE	1808	
		14	AE	9F	001F5	PUSHAB	CODE		
			54	DD	001F8	PUSHL	TAG		
00000000G	00		03	FB	001FA	CALLS	#3, DBG\$STA_SYMTYPE		
			7E	D4	00201	CLRL	-(SP)	1809	
		10	AE	DD	00203	PUSHL	TYPE		
		18	AE	DD	00206	PUSHL	CODE		
			54	DD	00209	PUSHL	TAG		
			06	DD	0020B	PUSHL	#6		

00000000G	00	04	AC	DD	0020D	MOVL	PRM_DESC, R2	
	7E	14	52	DD	00211	PUSHL	R2	
		7A	06	FB	00213	PUSHAB	#6, DBG\$BUILD_PRIMARY_SUBNODE	1810
			8F	9A	0021D	MOVZBL	VAL_DESC	
00000000G	00		E2	DD	00221	PUSHL	#122, -(SP)	
	50	14	03	FB	00223	CALLS	#3, DBG\$PRIM_TO_VAL	
	54	20	AE	DD	0022A	MOVL	VAL_DESC, R0	1811
	6E	18	A0	DD	0022E	MOVL	32(R0), TAG	
			B2	OF	00232	REMQUE	@24(R2), DUMMY	1812
00000000G	00		57	DD	00236	PUSHL	MARK	1813
	52	1C	01	FB	00238	CALLS	#1, DBG\$POP_TEMPMEM	
			AE	DD	0023F	MOVL	TYPEID, R2	1814
			52	DD	00243	PUSHL	R2	
			54	DD	00245	PUSHL	TAG	
00000000G	00		02	FB	00247	CALLS	#2, DBG\$STA_VARIANT_SELECT	
			50	D5	0024E	TSTL	VARIANT	
			03	12	00250	BNEQ	36\$	
			FDDF	31	00252	BRW	4\$	
0A	A3		10	88	00255	BISB2	#16, 10(SUB_NODE)	1815
18	A3		01	B0	00259	MOVW	#1, 24(SUB_NODE)	1816
		0C	A3	D4	0025D	CLRL	12(SUB_NODE)	1817
1C	A3	10	A2	DD	00260	MOVL	16(R2), 28(SUB_NODE)	1818
1A	A3	04	A0	B0	00265	MOVW	4(VARIANT), 26(SUB_NODE)	1819
20	A3	08	A0	9E	0026A	MOVAB	8(R0), 32(SUB_NODE)	1820
24	A3		60	DD	0026F	MOVL	(VARIANT), 36(SUB_NODE)	1821
	50	04	AC	DD	00273	MOVL	PRM_DESC, R0	1822
18	B0		63	0E	00277	INSQUE	(SUB_NODE), @24(R0)	
08	AE	1A	A3	3C	0027B	MOVZWL	26(SUB_NODE), N_COMPS	1824
04	AE	20	A3	DD	00280	MOVL	32(SUB_NODE), S_VECT	1825
	05		5A	E9	00285	BLBC	COMP_FLAG, 39\$	1827
18	A3	08	AE	B0	00288	MOVW	N_COMPS, 24(SUB_NODE)	
	50	18	A3	3C	0028D	MOVZWL	24(SUB_NODE), R0	1828
	50	04	BE40	DE	00291	MOVAL	@S_VECT(R0), R0	
	55	FC	A0	DD	00296	MOVL	-4(R0), SYMID	
		18	AE	9F	0029A	PUSHAB	KIND	1829
			55	DD	0029D	PUSHL	SYMID	
00000000G	00		02	FB	0029F	CALLS	#2, DBG\$STA_SYMKIND	
	0B	18	AE	D1	002A6	CMPL	KIND, #11	1830
			0C	12	002AA	BNEQ	40\$	
	20		13	DD	002AC	MOVL	#19, FCODE	1833
1C	AE		55	DD	002B0	MOVL	SYMID, TYPEID	1834
			55	D4	002B4	CLRL	SYMID	1835
			0F	11	002B6	BRB	42\$	1830
		1C	AE	9F	002B8	PUSHAB	TYPEID	1838
		24	AE	9F	002BB	PUSHAB	FCODE	
			55	DD	002BE	PUSHL	SYMID	
00000000G	00		03	FB	002C0	CALLS	#3, DBG\$STA_SYMTYPE	
0A	A3		01	88	002C7	BISB2	#1, 10(SUB_NODE)	1844
			7E	D4	002CB	CLRL	-(SP)	1845
		20	AE	DD	002CD	PUSHL	TYPEID	
		28	AE	DD	002D0	PUSHL	FCODE	
			55	DD	002D3	PUSHL	SYMID	
	52	28	AE	DD	002D5	PUSHL	KIND	
		04	AC	DD	002D8	MOVL	PRM_DESC, R2	
00000000G	00		52	DD	002DC	PUSHL	R2	
			06	FB	002DE	CALLS	#6, DBG\$BUILD_PRIMARY_SUBNODE	

DBGLEVEL3  
V04-000

B 11  
16-Sep-1984 01:30:26 VAX-11 Bliss-32 V4.0-742  
14-Sep-1984 12:17:02 [DEBUG.SRC]DBGLEVEL3.B32;1

Page 57  
(11)

53	18	A2	D0	002E5	MOVL	24(R2), SUB_NODE	1846
5A	08	AC	90	002E9	MOVB	DIRECTION, COMP_FLAG	1847
		FE70	31	002ED	BRW	27\$	1756
	10	A3	D5	002F0	TSTL	16(SUB_NODE)	1850
		15	13	002F3	BEQL	44\$	
	24	AE	9F	002F5	PUSHAB	SYM_NAME	1851
00000000G	00	10	A3	DD	PUSHL	16(SUB_NODE)	
		02	FB	002FB	CALLS	#2, DBG\$STA_SYMNAME	
	24	BE	95	00302	TSTB	@SYM_NAME	1852
		03	12	00305	BNEQ	44\$	
		FD2A	31	00307	BRW	4\$	
50		01	D0	0030A	MOVL	#1, R0	1855
			04	0030D	RET		
		50	D4	0030E	CLRL	R0	1856
			04	00310	RET		

; Routine Size: 785 bytes, Routine Base: DBG\$CODE + 0C10

```

1735 1857 1 ROUTINE PRIMARY_ORDER( PRIM_1: REF DBG$PRIMARY, PRIM_2: REF DBG$PRIMARY) =
1736 1858 1
1737 1859 1 FUNCTION
1738 1860 1 -----
1739 1861 1
1740 1862 1 INPUTS
1741 1863 1 -----
1742 1864 1
1743 1865 1 OUTPUTS
1744 1866 1 -----
1745 1867 1
1746 1868 1
1747 1869 2 BEGIN
1748 1870 2 LOCAL
1749 1871 2     NODE_1 : REF DBG$PRIM_NODE,
1750 1872 2     NODE_2 : REF DBG$PRIM_NODE,
1751 1873 2     VALUE_1,
1752 1874 2     VALUE_2;
1753 1875 2
1754 1876 2     NODE_1 = .PRIM_1[DBG$L_PRIM_FLINK];
1755 1877 2     NODE_2 = .PRIM_2[DBG$L_PRIM_FLINK];
1756 1878 2
1757 1879 2 WHILE TRUE DO
1758 1880 3 BEGIN
1759 1881 3
1760 1882 3     SELECTONE .NODE_1[DBG$B_PNODE_FCODE] OF
1761 1883 3     SET
1762 1884 3     [RST$K_TYPE_RECORD, RST$K_TYPE_VARIANT]:
1763 1885 4 BEGIN
1764 1886 4     IF .NODE_1[DBG$B_PNODE_FCODE] EQL RST$K_TYPE_VARIANT THEN
1765 1887 4     IF .NODE_1[DBG$B_PNVAR_DSTPTR] NEQ .NODE_2[DBG$B_PNVAR_DSTPTR]
1766 1888 4     THEN SIGNAL( DBG$EXARANGE);
1767 1889 4     VALUE_1 = .NODE_1[DBG$B_PNREC_INDEX];
1768 1890 4     VALUE_2 = .NODE_2[DBG$B_PNREC_INDEX];
1769 1891 4     IF .VALUE_1 LSS .VALUE_2 THEN RETURN -1;
1770 1892 4     IF .VALUE_1 GTR .VALUE_2 THEN RETURN +1;
1771 1893 3     END;
1772 1894 3
1773 1895 3 [RST$K_TYPE_ARRAY]:
1774 1896 4 BEGIN
1775 1897 4 LOCAL
1776 1898 4     SUBS_1 : REF DBG$PRIM_NODE_SUBS,
1777 1899 4     SUBS_2 : REF DBG$PRIM_NODE_SUBS;
1778 1900 4
1779 1901 4     IF .NODE_1[DBG$B_PNARR_SUBCNT] NEQ .NODE_2[DBG$B_PNARR_SUBCNT]
1780 1902 4     THEN SIGNAL( DBG$EXARANGE);
1781 1903 4
1782 1904 4     SUBS_1 = NODE_1[DBG$A_PNARR_SVECTOR];
1783 1905 4     SUBS_2 = NODE_2[DBG$A_PNARR_SVECTOR];
1784 1906 4
1785 1907 4     INCR DIMENSION FROM 0 TO .NODE_1[DBG$B_PNARR_DIMCNT]-1 DO
1786 1908 5 BEGIN
1787 1909 5 LOCAL D;
1788 1910 6     D = (IF .NODE_1[DBG$B_PNARR_COLUMN]
1789 1911 6     THEN .NODE_1[DBG$B_PNARR_DIMCNT] - .DIMENSION - 1
1790 1912 5     ELSE .DIMENSION);
1791 1913 5

```



```

1792 1914 5
1793 1915 5
1794 1916 5
1795 1917 5
1796 1918 5
1797 1919 5
1798 1920 5
1799 1921 5
1800 1922 5
1801 1923 5
1802 1924 5
1803 1925 5
1804 1926 5
1805 1927 5
1806 1928 5
1807 1929 5

```

```

VALUE_1 = .SUBS_1[.D,DBG$PNSUB_SVALUE] - .SUBS_1[.D,DBG$PNSUB_LBOUND];
VALUE_2 = .SUBS_2[.D,DBG$PNSUB_SVALUE] - .SUBS_2[.D,DBG$PNSUB_LBOUND];
IF .VALUE_1 LSS .VALUE_2 THEN RETURN -1;
IF .VALUE_1 GTR .VALUE_2 THEN RETURN +1;
END;
END;
[OTHERWISE]:
EXITLOOP;
TES;
NODE_1 = .NODE_1[DBG$PNODE_FLINK];
NODE_2 = .NODE_2[DBG$PNODE_FLINK];
END;
RETURN 0;
END;
! End of primary_order

```

03FC 00000 PRIMARY_ORDER:						
59	00000000G	00	9E 00002	WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	1857
50	04	AC	D0 00009	MOVAB	LIB\$SIGNAL, R9	1876
52	14	A0	D0 0000D	MOVL	PRIM 1, R0	
50	08	AC	D0 00011	MOVL	20(R0), NODE_1	1877
53	14	A0	D0 00015	MOVL	PRIM 2, R0	
50	09	A2	9A 00019 1\$:	MOVZBL	20(R0), NODE_2	1882
07		50	91 0001D		9(NODE_1), R0	1884
		05	13 00020	CMPB	R0, #7	
13		50	91 00022	BEQL	2\$	
		26	12 00025	CMPB	R0, #19	
13		50	91 00027 2\$:	BNEQ	4\$	
		10	12 0002A	CMPB	R0, #19	1886
24	A3 24	A2	D1 0002C	BNEQ	3\$	
		09	13 00031	CMPL	36(NODE_1), 36(NODE_2)	1887
	00028190	8F	DD 00033	BEQL	3\$	1888
69		01	FB 00039	PUSHL	#164240	
57	18	A2	3C 0003C 3\$:	CALLS	#1, LIB\$SIGNAL	1889
56	18	A3	3C 00040	MOVZWL	24(NODE_1), VALUE_1	1890
56		57	D1 00044	MOVZWL	24(NODE_2), VALUE_2	1891
		5B	19 00047	CMPL	VALUE_1, VALUE_2	
		67	15 00049	BLSS	9\$	
		5D	11 0004B	BLEQ	13\$	1892
01		50	91 0004D 4\$:	BRB	11\$	
		69	12 00050	CMPB	R0, #1	1895
1F	A3 1F	A2	91 00052	BNEQ	14\$	
		09	13 00057	CMPB	31(NODE_1), 31(NODE_2)	1901
	00028190	8F	DD 00059	BEQL	5\$	
69		01	FB 0005F	PUSHL	#164240	1902
51	28	A2	9E 00062 5\$:	CALLS	#1, LIB\$SIGNAL	
50	28	A3	9E 00066	MOVAB	40(R2), SUBS_1	1904
58	1B	A2	9A 0006A	MOVAB	40(R3), SUBS_2	1905
55		01	CE 0006E	MOVZBL	27(NODE_1), R8	1907
		3B	11 00071	MNEGL	#1, DIMENSION	1915
0B	0A A2	01	E1 00073 6\$:	BRB	12\$	
				BBC	#1, 10(NODE_1), 7\$	1910

	54	18	A2	9A	00078	MOVZBL	27(NODE 1), R4	1911
	54		55	C2	0007C	SUBL2	DIMENSION, R4	
			54	D7	0007F	DECL	D	
			03	11	00081	BRB	8\$	
	54		55	D0	00083	7\$: MOVL	DIMENSION, D	1912
	54		14	C4	00086	8\$: MULL2	#20, R4	1914
			6441	9F	00089	PUSHAB	(R4)[SUBS 1]	
57	9E	08	A441	9F	0008C	PUSHAB	8(R4)[SUBS 1]	
			9E	C3	00090	SUBL3	@(SP)+, @(SP)+, VALUE_1	
			6440	9F	00094	PUSHAB	(R4)[SUBS 2]	1915
56	9E	08	A440	9F	00097	PUSHAB	8(R4)[SUBS 2]	
			9E	C3	0009B	SUBL3	@(SP)+, @(SP)+, VALUE_2	
	56		57	D1	0009F	CMPL	VALUE_1, VALUE_2	1916
			04	18	000A2	BGEQ	10\$	
	50		01	CE	000A4	9\$: MNEGL	#1, R0	
				04	000A7	RET		
			04	15	000AB	10\$: BLEQ	12\$	1917
	50		01	D0	000AA	11\$: MOVL	#1, R0	
				04	000AD	RET		
C1	55		58	F2	000AE	12\$: AOBLSS	R8, DIMENSION, 6\$	1907
	52		62	D0	000B2	13\$: MOVL	(NODE_1), NODE_1	1925
	53		63	D0	000B5	MOVL	(NODE_2), NODE_2	1926
			FF5E	31	000B8	BRW	1\$	1879
			50	D4	000BB	14\$: CLRL	R0	1928
				04	000BD	RET		1929

; Routine Size: 190 bytes, Routine Base: DBG\$CODE + 0F21

```

1809 1930 ROUTINE CHECK_TEXT_DESCRIPTOR(VAL_DESC: REF DBG$VALDESC) =
1810 1931
1811 1932 FUNCTION
1812 1933 -----
1813 1934
1814 1935 INPUTS
1815 1936 -----
1816 1937
1817 1938 OUTPUTS
1818 1939 -----
1819 1940
1820 1941
1821 1942 BEGIN
1822 1943
1823 1944 BUILTIN
1824 1945 PROBER;
1825 1946
1826 1947 BIND VMS_DESC = VAL_DESC[DBG$A_VALUE_VMSDESC]: DBG$STG_DESC;
1827 1948
1828 1949
1829 1950
1830 1951 IF (.VAL_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS) THEN SIGNAL(DBG$UNALIGNED);
1831 1952
1832 1953 IF NOT PROBER(%REF(0),%REF(8),.VAL_DESC[DBG$L_VALUE_POINTER])
1833 1954 THEN SIGNAL(DBG$NOACCESSR,1,.VAL_DESC[DBG$L_VALUE_POINTER]);
1834 1955
1835 1956 CH$MOVE(8,.VAL_DESC[DBG$L_VALUE_POINTER],VAL_DESC[DBG$A_VALUE_VMSDESC]);
1836 1957
1837 1958 IF (.VMS_DESC[DSC$B_DTYPE] EQL 0) AND (.VMS_DESC[DSC$B_CLASS] EQL 0)
1838 1959 THEN
1839 1960 BEGIN
1840 1961 VMS_DESC[DSC$B_DTYPE] = DSC$K_DTYPE_T;
1841 1962 VMS_DESC[DSC$B_CLASS] = DSC$K_CLASS_S;
1842 1963 RETURN TRUE;
1843 1964 END;
1844 1965
1845 1966 IF (.VMS_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_D)
1846 1967 THEN VMS_DESC[DSC$B_CLASS] = DSC$K_CLASS_S;
1847 1968
1848 1969 IF (.VMS_DESC[DSC$B_CLASS] EQL DSC$K_CLASS_VS)
1849 1970 AND (.VMS_DESC[DSC$B_DTYPE] EQL DSC$K_DTYPE_T)
1850 1971 THEN VMS_DESC[DSC$B_DTYPE] = DSC$K_DTYPE_VT;
1851 1972
1852 1973 IF (.VMS_DESC[DSC$B_CLASS] NEQ DSC$K_CLASS_VS)
1853 1974 AND (.VMS_DESC[DSC$B_CLASS] NEQ DSC$K_CLASS_S) THEN RETURN FALSE;
1854 1975
1855 1976 SELECTONE .VMS_DESC[DSC$B_DTYPE] OF
1856 1977 SET
1857 1978 [DSC$K_DTYPE_T] : VMS_DESC[DSC$B_CLASS] = DSC$K_CLASS_S;
1858 1979
1859 1980 [DSC$K_DTYPE_AZ,
1860 1981 DSC$K_DTYPE_AC,
1861 1982 DSC$K_DTYPE_VT]: VMS_DESC[DSC$B_CLASS] = DSC$K_CLASS_VS;
1862 1983
1863 1984 [OTHERWISE]: RETURN FALSE;
1864 1985
1865 1986 TES;

```

```
! End of routine check_text_descriptor
```

00FC 00000 CHECK\_TEXT\_DESCRIPTOR:

COPY 00000 CHECK-TEXT DESCRIPTION:									
	57	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7	1930	
	52		04	AC	D0	00009	MOVAB	LIBSSIGNAL, R7	1947
	56		14	A2	9E	0000D	MOVL	VAL_DESC, R2	1951
	0D		03	A6	91	00011	MOVAB	20(R2), R6	
				09	12	00015	CMPB	3(R6), #13	
		00028D08		8F	DD	00017	BNEQ	1\$	
18	B2	67		01	FB	0001D	PUSHL	#167176	
		08		00	0C	00020	CALLS	#1, LIBSSIGNAL	1953
				0E	12	00025	PROBER	#0, #8, @24(R2)	
			18	A2	DD	00027	BNEQ	2\$	1954
				01	DD	0002A	PUSHL	24(R2)	
		00028228		8F	DD	0002C	PUSHL	#1	
	67			03	FB	00032	PUSHL	#164392	
66	TB	82		08	28	00035	CALLS	#3, LIBSSIGNAL	
		51	02	A6	9E	0003A	MOVC3	#8, @24(R2), (R6)	1956
				61	95	0003E	MOVAB	2(R6), R1	1958
				0E	12	00040	TSTB	(R1)	
			03	A6	95	00042	BNEQ	3\$	
				09	12	00045	TSTB	3(R6)	
	61			0E	90	00047	BNEQ	3\$	1961
	03	A6		01	90	0004A	MOVB	#14, (R1)	1962
				3A	11	0004E	MOVB	#1, 3(R6)	1963
	50		03	A6	9E	00050	BRB	8\$	1966
	02			60	91	00054	MOVAB	3(R6), R0	
				03	12	00057	CMPB	(R0), #2	
	60			01	90	00059	BNEQ	4\$	
	0B			60	91	0005C	MOVB	#1, (R0)	1967
				08	12	0005F	CMPB	(R0), #11	1969
	0E			61	91	00061	BNEQ	5\$	
				03	12	00064	CMPB	(R1), #14	1970
	61			25	90	00066	BNEQ	5\$	
	0B			60	91	00069	MOVB	#37, (R1)	1971
				05	13	0006C	CMPB	(R0), #11	1973
	01			60	91	0006E	BEQL	6\$	
				1B	12	00071	CMPB	(R0), #1	1974
	0E			61	91	00073	BNEQ	9\$	
				05	12	00076	CMPB	(R1), #14	1978
	60			01	90	00078	BNEQ	7\$	
				0D	11	0007B	MOVB	#1, (R0)	
	25			61	91	0007D	BRB	8\$	
				0C	1F	00080	CMPB	(R1), #37	1980
	27			61	91	00082	BLSSU	9\$	
				07	1A	00085	CMPB	(R1), #39	
	60			0B	90	00087	BGTRU	9\$	
	50			01	D0	0008A	MOVB	#11, (R0)	1982
					04	0008D	MOVL	#1, R0	1988
				50	D4	0008E	RET	R0	1989
							CLRL	R0	



DBGLEVEL3  
V04-000

M 11  
16-Sep-1984 01:30:26  
14-Sep-1984 12:17:02

VAX-11 B11gs-32 V4.0-742  
[DEBUG.SRC]DBGLEVEL3.B32;1

Page 63  
(13)

04 00090 RET

; Routine Size: 145 bytes, Routine Base: DBG\$CODE + 0FDF

```
1870 1990 1 ROUTINE FIX_UP_LENGTH(VMS_DESC: REF DBG$STG_DESC) =
1871 1991 1
1872 1992 1 FUNCTION
1873 1993 1 -----
1874 1994 1
1875 1995 1 INPUTS
1876 1996 1 -----
1877 1997 1
1878 1998 1 OUTPUTS
1879 1999 1 -----
1880 2000 1
1881 2001 1
1882 2002 1 BEGIN
1883 2003 1
1884 2004 1 LOCAL
1885 2005 1 SIZE,
1886 2006 1 BASE: REF BLOCK[,BYTE];
1887 2007 1
1888 2008 1
1889 2009 1
1890 2010 1 BASE = .VMS_DESC[DSC$A_POINTER];
1891 2011 1 SELECTONE .VMS_DESC[DSC$B_DTYPE] OF
1892 2012 1 SET
1893 2013 1
1894 2014 1 [DSC$K_DTYPE_VT]:
1895 2015 1 BEGIN
1896 2016 1 BUILTIN PROBER;
1897 2017 1 IF NOT PROBER(%REF(0),%REF(2),.BASE)
1898 2018 1 THEN SIGNAL(DBG$NOACCESSR,1,.BASE);
1899 2019 1 SIZE = .BASE[0,0,16,0];
1900 2020 1 END;
1901 2021 1
1902 2022 1 [DSC$K_DTYPE_AC]:
1903 2023 1 BEGIN
1904 2024 1 BUILTIN PROBER;
1905 2025 1 IF NOT PROBER(%REF(0),%REF(1),.BASE)
1906 2026 1 THEN SIGNAL(DBG$NOACCESSR,1,.BASE);
1907 2027 1 SIZE = .BASE[0,0,8,0];
1908 2028 1 END;
1909 2029 1
1910 2030 1 [DSC$K_DTYPE_AZ]:
1911 2031 1 BEGIN
1912 2032 1 BUILTIN LOCC,PROBER;
1913 2033 1 LOCAL ADDR;
1914 2034 1 IF NOT PROBER(%REF(0),%REF(4),.BASE)
1915 2035 1 THEN SIGNAL(DBG$NOACCESSR,1,.BASE);
1916 2036 1 LOCC(%REF(0),%REF(2048),.BASE,%ADDR);
1917 2037 1 SIZE = .ADDR - .BASE;
1918 2038 1 END;
1919 2039 1
1920 2040 1 [OTHERWISE]:
1921 2041 1 SIZE = .VMS_DESC[DSC$W_LENGTH];
1922 2042 1
1923 2043 1 TES;
1924 2044 1
1925 2045 1 RETURN .SIZE;
1926 2046 1
```

: 1927

2047 1 END;

! End of routine fix\_up\_length

		001C 00000	FIX_UP_LENGTH:			
	54	00000000G	00 9E 00002	.WORD	Save R2,R3,R4	1990
	50	04	AC D0 00009	MOVAB	LIB\$SIGNAL, R4	
	52	04	A0 D0 0000D	MOVL	VMS DESC, R0	2010
	51	02	A0 9A 00011	MOVL	4(R0), BASE	
	25		51 91 00015	MOVZBL	2(R0), R1	2011
			18 12 00018	CMPB	R1, #37	2014
62	02		00 0C 0001A	BNEQ	2\$	
			0D 12 0001E	PROBER	#0, #2, (BASE)	2017
			52 DD 00020	BNEQ	1\$	
			01 DD 00022	PUSHL	BASE	2018
		00028228	8F DD 00024	PUSHL	#1	
64			03 FB 0002A	PUSHL	#164392	
53			62 3C 0002D 1\$:	CALLS	#3, LIB\$SIGNAL	
			44 11 00030	MOVZWL	(BASE), SIZE	2019
			51 91 00032 2\$:	BRB	7\$	2011
26			18 12 00035	CMPB	R1, #38	2022
62	01		00 0C 00037	BNEQ	4\$	
			0D 12 0003B	PROBER	#0, #1, (BASE)	2025
			52 DD 0003D	BNEQ	3\$	
			01 DD 0003F	PUSHL	BASE	2026
		00028228	8F DD 00041	PUSHL	#1	
64			03 FB 00047	PUSHL	#164392	
53			62 9A 0004A 3\$:	CALLS	#3, LIB\$SIGNAL	
			27 11 0004D	MOVZBL	(BASE), SIZE	2027
			51 91 0004F 4\$:	BRB	7\$	2011
			1F 12 00052	CMPB	R1, #39	2030
62	04		00 0C 00054	BNEQ	6\$	
			0D 12 00058	PROBER	#0, #4, (BASE)	2034
			52 DD 0005A	BNEQ	5\$	
			01 DD 0005C	PUSHL	BASE	2035
		00028228	8F DD 0005E	PUSHL	#1	
62	0800		03 FB 00064	PUSHL	#164392	
53			00 3A 00067 5\$:	CALLS	#3, LIB\$SIGNAL	
			52 C3 0006D	LOCC	#0, #2048, (BASE)	2036
			03 11 00071	SUBL3	BASE, ADDR, SIZE	2037
			60 3C 00073 6\$:	BRB	7\$	2011
			53 D0 00076 7\$:	MOVZWL	(R0), SIZE	2041
			04 00079	MOVL	SIZE, R0	2045
				RET		2047

: Routine Size: 122 bytes, Routine Base: DBG\$CODE + 1070

: 1928  
: 19292048 1  
2049 0 END ELUDOM

.EXTRN LIB\$SIGNAL

# PSECT SUMMARY

Name	Bytes	Attributes							
DBG\$OWN	4	NOVEC,	WRT,	RD	,NOEXE,	NOSHR,	LCL,	REL,	CON,
DBG\$CODE	4330	NOVEC,	NOWRT,	RD	,EXE,	SHR,	LCL,	REL,	CON,
DBG\$PLIT	74	NOVEC,	NOWRT,	RD	,EXE,	SHR,	LCL,	REL,	CON,
									PIC,ALIGN(2)
									PIC,ALIGN(0)
									PIC,ALIGN(0)

# Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
-\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	20	0	1000	00:01.9
-\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	1	3	7	00:00.1
-\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	190	12	97	00:02.0
-\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	2	0	31	00:00.3
-\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	11	2	22	00:00.3
-\$255\$DUA28:[DEBUG.OBJ]DBGGEN.L32;1	150	0	0	12	00:00.3

# COMMAND QUALIFIERS

```

:
: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$;DBGLEVEL3/OBJ=OBJ$;DBGLEVEL3 MSRC$;DBGLEVEL3/UPDATE=(ENH$;DBGLEVEL3)
:
: Size:          4330 code + 78 data bytes
: Run Time:      01:11.8
: Elapsed Time:  03:41.8
: Lines/CPU Min: 1713
: Lexemes/CPU-Min: 14910
: Memory Used:   362 pages
: Compilation Complete

```



0085 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

